Desenvolvimento web com

PHP e MySQL

EVALDO JUNIOR BENTO

Agradecimentos

A vida não é uma sequência de acontecimentos aleatórios. Muita coisa teve que acontecer em uma certa ordem para que este livro fosse possível, desde uma longínqua oportunidade de fazer o primeiro curso de informática, passando por curso técnico, faculdade, grupos de estudo, palestras, até a chance de trabalhar com pessoas que me fizeram evoluir. Por isso agradeço a Deus por ter me dado as ferramentas e o discernimento necessários para encontrar os caminhos que me trouxeram até este livro.

Este livro não seria uma realidade sem o apoio da minha amada esposa *Cássia Luz*. Quando eu falei para ela da oportunidade de escrever um livro, ela disse **você tem que escrever um livro!** e isso me motivou bastante a encarar essa tarefa tão difícil. Obrigado, Cassinha! (E obrigado também pela ideia da ilustração da capa, ficou demais!)

Agradeço também aos meus pais, Silvana e Evaldo, meus irmãos, Jenner e Antonio Paulo, e à sra. Creuza e ao sr. Mário, tios da Cássia, que me desculparam por não comparecer aos almoços e jantares em família porque estava escrevendo só mais um pedaço do livro.

Na verdade, eu sempre quis escrever um livro sobre programação e até cheguei começar alguns rascunhos, mas acabei deixando todos de lado. Até que um dia o Caio Ribeiro Pereira, que estava terminando seu livro de Node.js, me perguntou se eu não queria escrever um livro de Python e me apresentou ao Paulo Silveira da Caelum/Casa do Código. Depois disso trocamos alguns e-mails e comecei a escrever este livro de PHP e MySQL, finalmente colocando no "papel" o que pensei em fazer por muito tempo. Obrigado, Caio! E obrigado, Paulo!

Agradeço também aos meus alunos da Unimonte que pacientemente aguardaram os quase seis meses de escrita deste livro. Não falei para vocês que eu terminaria logo?

Obrigado também ao pessoal que leu alguns rascunhos e me deu ideias do que

poderia ser abordado.

Não posso esquecer do pessoal do GCCSD que foi bastante incentivador quando eu ainda dava passos pequenos em TI.

Sobre o autor

Evaldo Junior Bento trabalha com TI desde 2004. É desenvolvedor web com foco em boas práticas e padrões de desenvolvimento utilizando PHP como sua principal linguagem desde 2008. É professor universitário, ministrando disciplinas relacionadas a desenvolvimento de software e também palestrante em eventos relacionados a software livre e desenvolvimento de software. Possui formação em Processamento de Dados pela Fatec e Pós Graduação em Gestão Estratégica de TI. Mantém um blog sobre desenvolvimento e tecnologia em http://evaldojunior.com.br/blog/ e projetos open source no GitHub em https://github.com/InFog.

Prefácio

Muita gente que pensa em Web lembra logo de HTML, CSS e JavaScript. Claro, são as linguagens fundamentais dos navegadores. Mas elas só contam metade da história. Muita coisa precisa acontecer do outro lado da conexão, nos servidores. E, se você quer programar seriamente na Web, vai precisar dominar uma linguagem de programação que rode no servidor.

Nesse cenário, o PHP é a melhor escolha se você quer evoluir suas páginas estáticas HTML para páginas dinâmicas e sistemas complexos. Há muitas opções de linguagens de programação e de bancos de dados, mas a dupla PHP e MySQL é das mais importantes no mercado Web atual, com aplicações em sites de conteúdo, comércio eletrônico e até sistemas grandes como Facebook.

Esse livro é obrigatório para quem quer começar com PHP e MySQL. O Evaldo é um excelente autor com grande experiência, e conseguiu chegar em um livro fácil de ler e acompanhar, com uma escrita dinâmica e vibrante. Segue o modelo de livros objetivos da Casa do Código e fará você entrar nesse mercado rapidamente e com bastante conhecimento.

Uma boa leitura, bons estudos e bem-vindo ao imenso e importante mundo do PHP.

Sérgio Lopes

— Instrutor e desenvolvedor na Caelum, autor do livro "A Web Mobile" também da editora Casa do Código

http://sergiolopes.org

Sumário

Capítulo 1

Introdução

1.1 Ganhando a web 2.0 com páginas dinâmicas

Imagine a internet na qual você pode apenas consumir conteúdos, como se fosse um jornal, uma revista, ou ainda, um programa na televisão. Chato, né? Mas quando se aprende as linguagens da web, como HTML e CSS, é isso que se aprende, pois usando apenas HTML podemos montar sites que são como revistas e servem apenas para leitura, sem permitir interação com os internautas.

O segredo da famosa **web 2.0** é a capacidade de interação entre as pessoas e os serviços online. Mas, para que esta interação seja possível, é necessário que os sites sejam capazes de receber informações dos internautas e também de exibir conteúdos personalizados para cada um ou de mudar seu conteúdo automaticamente, sem que o desenvolvedor precise criar um novo HTML para isso.

Estes dois tipos de sites são chamados de estático e dinâmico, respectivamente.

1.2 O navegador e a exibição de páginas web estáticas

Você já parou para pensar em tudo o que acontece quando você digita um endereço em seu navegador web? A história toda é mais ou menos assim:

- O navegador vai até o servidor que responde no endereço solicitado e pede a página solicitada.
- O servidor verifica se o endereço existe e se a página também existe em seu sistema de arquivos e então retorna o arquivo para o navegador.
- Após receber o arquivo HTML, o navegador começa o trabalho de renderização, para exibir a página para o usuário. É neste momento que o navegador também requisita arquivos de estilos (css), imagens e outros arquivos necessários para a exibição da página.

Quando se desenvolve páginas estáticas, este é basicamente todo o processo necessário para que o navegador exiba a página para o usuário. Chamamos de **estáticas** as páginas web que não mudam seu conteúdo, mesmo em uma nova requisição ao servidor.

1.3 E como funciona uma página dinâmica?

O processo para páginas dinâmicas é muito parecido com o das páginas estáticas. A diferença é que a página será processada no servidor antes de ser enviada para o usuário. Este processamento no servidor é usado para alterar dinamicamente o conteúdo de uma página, seja ele HTML, CSS, imagens ou outros formatos.

Pense, por exemplo, em um site de um jornal. Em geral, este tipo de site contém algumas áreas destinadas às notícias de destaque, outras áreas para notícias gerais e ainda outras áreas para outros fins. Quando o navegador solicita a página para o servidor, ele irá montar o conteúdo antes de enviar para o navegador. Este conteúdo pode ser conseguido de algumas fontes, mas a mais comum é um banco de dados, onde, neste caso, as notícias ficam armazenadas para serem exibidas nas páginas quando necessário.

1.4 E onde entra o PHP e o MySQL?

PHP é uma ferramenta que possibilita o pré-processamento de páginas HTML. Dessa forma, PHP consegue alterar o conteúdo de uma página, antes de enviá-la para o navegador. Além disso, PHP também permite capturar entradas de dados do usuário, como formulários e outras formas de interação.

Já o MySQL é o banco de dados no qual guardamos informações em estruturas no estilo de tabelas, sendo que cada linha da tabela é um novo registro. É em bancos como o MySQL que os sites de notícias, redes sociais etc., guardam suas informações para que depois sejam recuperadas e exibidas nas páginas.

A dupla PHP e MySQL se conhece há muitos anos e trabalha bem em equipe.

1.5 Mas, por que escolher PHP e MySQL?

Há alguns dias, ao final de uma aula na faculdade, um aluno veio até mim e perguntou por que as empresas escolhem PHP e MySQL para desenvolver seus sites e até mesmo seus sistemas. Ele me disse que existem linguagens superiores ao PHP e bancos que são tidos como melhores que o MySQL.

Responder a esta questão não é fácil, pois existem diversos motivos para escolher esta ou aquela tecnologia. No caso da dupla PHP e MySQL, alguns motivos são:

- PHP nasceu para a web e sua integração com servidores web é simples.
- PHP tem uma curva de aprendizado suave, comparada a outras linguagens.
- PHP e MySQL são tecnologias livres.
- É fácil encontrar serviços de hospedagem que oferecem PHP e MySQL.
- Serviços de hospedagem PHP e MySQL são mais baratos que serviços semelhantes para outras tecnologias.
- MySQL é leve e rápido, mesmo para quantidades razoavelmente grandes de dados.

1.6 O que vou precisar para acompanhar este livro?

Para desenvolver software são necessárias algumas ferramentas. Neste livro, farei uso e indicarei apenas ferramentas em software livre, mas você pode usar ferramentas

que já conhece e com as quais se sinta confortável, apenas se certificando de fazer as devidas adaptações quando necessário.

No geral tudo o que será necessário é um computador com o ambiente WEB com PHP e MySQL, um bom editor de textos e um navegador WEB para testar as páginas que serão criadas.

Uma dica importante para quem busca aprender uma nova linguagem de programação, ou mesmo a primeira linguagem de programação, é reservar tempo para estudar e praticar bastante. Se você conseguir separar um certo tempo por dia e realmente se dedicar à leitura e prática dos exercícios propostos, rapidamente você se sentirá mais confortável com PHP e com o ambiente WEB, o que vai lhe dar conceitos gerais para desenvolvimento de páginas dinâmicas até mesmo usando outras linguagens.

Ou seja, um dos requisitos para o estudo será mesmo o tempo e, quanto mais tempo você conseguir dedicar aos estudos, mais conseguirá absorver novos conhecimentos e mais rápido conseguirá desenvolver suas aplicações.

1.7 Sobre este livro

A ideia central deste livro é oferecer a oportunidade de o leitor começar a desenvolver suas primeiras páginas dinâmicas utilizando a linguagem PHP associada ao banco de dados MySQL. Este livro apresenta uma experiência de aprendizado que pode (e deve) ser aplicada não somente ao PHP, mas também a quaisquer outras tecnologias para desenvolvimento de aplicações, sejam elas web ou não. Durante os capítulos, os exemplos são construídos aos poucos e alguns erros são encorajados, além de haver algumas reescritas e melhorias em códigos que já funcionam — mas que podem sofrer por não utilizarem técnicas que simplificam a lógica e garantem maior facilidade para futuras alterações.

Este livro não é um guia de referência para PHP e MySQL e, assim sendo, não apresenta listas de funções e bibliotecas disponíveis para estas tecnologias. O foco aqui é realmente um processo de aprendizado, através da construção gradual de aplicações e assimilação dos conceitos.

Estudantes de cursos relacionados a desenvolvimento de sistemas, curiosos estudando programação para web e hobistas podem se beneficiar grandemente do conteúdo deste livro. Porém, desenvolvedores mais avançados que desejam apenas um guia de referência para tirar aquela dúvida sobre uma função ou outra da linguagem podem não encontrar benefícios nestas páginas.

Web designers com experiência em HTML e CSS que desejam aprender a desenvolver para backend também podem se beneficiar bastante do conteúdo deste livro.

Mesmo sendo focado no iniciante, este livro busca trazer conteúdo atualizado com as práticas mais recentes do PHP e seu ambiente.

Sempre que estiver com alguma dúvida, não deixe de perguntar na lista de discussão do livro. Ela está em https://groups.google.com/forum/#!forum/phpemysqlcasadocodigo.

Os exemplos de código usados neste livro podem ser baixados no GitHub em https://github.com/InFog/phpmysql

Agora, acomode-se na cadeira e bons estudos!

Capítulo 2

O primeiro programa em PHP

Chegou o momento de escrever o nosso primeiro programa em PHP! Para isso, é necessário ter o PHP e algumas outras ferramentas instaladas no nosso computador.

Ainda não é hora de se preocupar com todas as ferramentas, pois precisamos realmente apenas do PHP. Mesmo assim iremos instalar um pacotão com praticamente tudo o que precisaremos durante o livro. Apenas não se preocupe com essas ferramentas agora, tudo bem?

2.1 Instalando o PHP

Uma ótima opção para instalar o PHP é o XAMPP. Este pacote contém tudo o que é preciso para começar a programar em PHP. Para instalar, acesse o site do XAMPP e clique na opção `XAMPP para Windows`, veja na imagem:

Figura 2.1: Site do XAMPP com link para o XAMPP para Windows

Ah, o site do XAMPP é este aqui: http://www.apachefriends.org/pt_br/xampp.html.

Na próxima página, clique no link `XAMPP` logo abaixo do título **Download**:

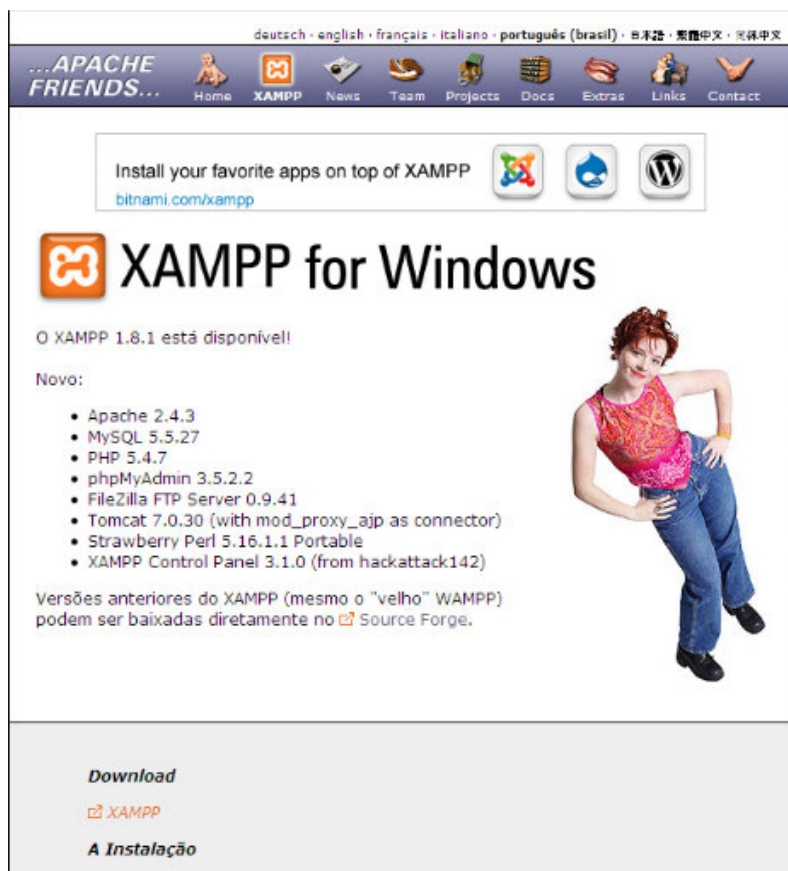



Figura 2.2: Opção para Download do XAMPP para Windows

Agora clique no link `Instalador`, você será direcionado para a página de download. O arquivo tem cerca de 100MB, aproveite para tomar um café enquanto baixa:

Figura 2.3: Download do instalador do XAMPP para Windows

Após finalizar o download, abra o instalador e use todas as opções padrão, apenas clicando em `Próximo` ou `Next` até finalizar a instalação.

Depois de instalar, clique em `Finalizar` ou `Finish`, deixando a opção de abrir o painel do XAMPP marcada:

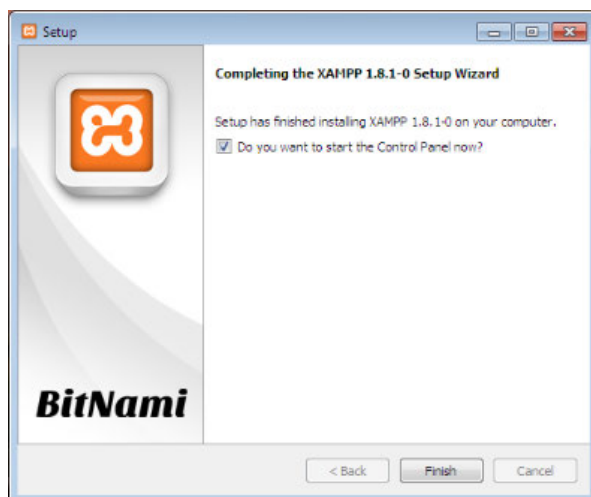



Figura 2.4: Finalizando o a instalação do XAMPP

No painel que abrir, clique no botão `start` apenas do serviço chamado **Apache**:

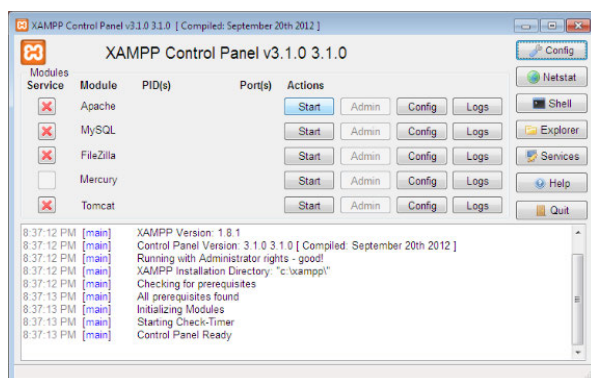



Figura 2.5: Painel do XAMPP, use a opção de iniciar o Apache

O Apache é o servidor web que será responsável por entregar as páginas quando o navegador solicitar.

Pode ser que seja necessário liberar o Apache no **firewall** do Windows:

Figura 2.6: Liberar o Apache no Firewall do Windows

O XAMPP será instalado na pasta `C:\xampp\`.

2.2 PHP no Linux

Se você usa Linux, use o gerenciador de pacotes da sua distribuição favorita para instalar o PHP. É bem fácil usar o **apt-get** no Debian e no Ubuntu ou o **yum** nos derivados do Fedora. Apenas procure pelos pacotes do PHP e do Apache e faça a instalação.

PHP no Debian/Ubuntu

Para instalar o PHP no Debian/Ubuntu, use o comando abaixo. Isso irá instalar o PHP, Apache e o MySQL e deixar tudo pronto para usar.

```
sudo aptitute install php5 php5-mysql apache2
libapache2-mod-php5 mysql-server
```

PHP no Mac OS X

Usuários do Mac OS X também podem usar o XAMPP, a página para download fica em http://www.apachefriends.org/en/xampp-macosx.html

Baixe o pacote DMG e siga as instruções de instalação.

Após a instalação, abra seu navegador e acesse o endereço **localhost**. Você deverá ver uma página de boas vindas do XAMPP, ou uma página dizendo que o Apache está funcionando:

Figura 2.7: Homepage do XAMPP instalado localmente

O que é o localhost?

Acabamos de acessar o endereço **localhost** através do navegador. Este é o endereço local do computador — ele corresponde sempre ao próprio computador. Após instalar o XAMPP, um servidor web também estará instalado no computador, por isso será possível acessá-lo através do navegador.

2.3 E vamos ao primeiro programa!

Certo, com o XAMPP instalado, é hora de escrever o primeiro programa em PHP! Abra um editor de textos e digite o seguinte código:

```
<?php

echo "Hoje é dia " . date('d/m/Y');
```

Agora, salve o arquivo como `C:\xampp\htdocs\programa.php` e acesse através do navegador o endereço `localhost/programa.php`, você deverá ver uma página assim:

```
Hoje é dia 26/05/2013
```

> **Onde salvar os arquivos no Linux e no Mac OS X?**
>
> Se você usa Debian ou Ubuntu, a pasta para salvar o arquivo é `/var/www`. Lembre-se de dar permissão para seu usuário criar arquivos nesta pasta.
>
> Usuários do Mac OS X que instalaram o XAMPP devem salvar os arquivos em `/Applications/XAMPP/htdocs/`.

> **O que é a pasta htdocs do XAMPP?**
>
> Esta pasta é a raiz do servidor Apache. É a partir dela que podemos acessar nossos arquivos dentro do Apache. Por exemplo, um arquivo chamado `pagina.php` dentro da pasta `htdocs` poderá ser acessado através do endereço `localhost/pagina.php`.
>
> Agora, se você criar uma pasta nova dentro de htdocs, por exemplo, a pasta **site** e dentro dela um arquivo `pagina.php`, o acesso será feito através do endereço `localhost/site/pagina.php`.

É claro que a data exibida no seu navegador será diferente, pois ela será gerada automaticamente pelo PHP. Vamos alterar um pouco este código, para enxergarmos melhor as mudanças acontecendo. Altere o arquivo `programa.php`, colocando o código a seguir:

```php
<?php

echo "Hoje é dia " . date('d/m/Y');
echo " agora são " . date('H:i:s');
```

Atualize a página no navegador, para isso use a opção de recarregar ou aperte o botão `F5`. Percebeu o que aconteceu? Agora ele também exibe a hora, os minutos e os segundos da hora da requisição. Experimente atualizar a página algumas vezes, veja que a cada requisição os dados serão alterados. E se você não atualizar mais o navegador? O que acontece?

NADA!

Isso mesmo, nada acontecerá. Mas, não era para ele continuar atualizando a hora?

Requisição, processamento e entrega

O que acontece é o seguinte: Quando o navegador pede uma página nova, ou atualiza uma sendo exibida, acontece uma **requisição**. O Apache recebe esta requisição e pede para o PHP fazer o **processamento**. É nesta hora que o PHP preenche os dados pegando a data e a hora do computador. Depois de processar tudo, o PHP devolve tudo para o Apache, que **entrega** para o navegador.

O navegador, então, exibe os dados já processados, e é por isso que a página não muda mais se uma nova requisição não for feita.

Alias, você sabia que PHP significa *PHP Hypertext Preprocessor*? Ou, em português: PHP Pré-processador de Hipertexto, sendo que hipertexto é o HTML. Legal, né?

2.4 A minha página está mostrando a hora errada!

A sua página poderá mostrar a hora errada, pois o PHP do XAMPP vem configurado com o a hora de outro país. Para corrigir isso, basta editar o arquivo de configuração do PHP dentro da instalação do XAMPP. O arquivo é o `php.ini` e, no XAMPP, ele fica em `c:\xampp\php\php.ini`. Neste arquivo procure pela linha que começa com `date.timezone =`. Na minha configuração ele veio assim:

```
date.timezone = Europe/Berlin
```

Veja que ele está com o horário de Berlim! Basta alterar para um horário brasileiro. No meu caso, ficou assim:

```
date.timezone = America/Sao_Paulo
```

Se você não estiver no horário de São Paulo (Brasília), você poderá pesquisar o seu timezone na lista de timezones da América que fica neste endereço: http://php.net/manual/en/timezones.america.php.

Usuários de Debian/Ubuntu vão encontrar o arquivo `php.ini` em `/etc/php5/apache2/php.ini`.

Após alterar e salvar o arquivo, será necessário reiniciar o Apache. Use o painel do XAMPP para parar e iniciar novamente o Apache. Se você usa Debian/Ubuntu, use o comando `sudo service apache2 restart`.

Qual editor de textos devo usar?

ATENÇÃO: Não utilize editores como o Word ou o Writer, pois estes editores gravam várias informações além do texto nos arquivos.

Existem diversos editores de texto que podem ser usados para editar seus códigos em PHP. No Windows, o mais simples deles é o bloco de notas, mas existem outras opções. A minha recomendação é o **Notepad++** um ótimo editor de textos que permite escrever em diversas linguagens, inclusive PHP e HTML. Se você ainda não usa um editor de textos para seus códigos, ou está começando agora, eu recomendo usar o Notepad++.

Para baixar o Notepad++ acesse http://notepad-plus-plus.org/ e clique na opção `Downloads`. Na próxima página, clique no link `Notepad++ Installer`. O instalador é simples, basta usar as opções padrão.

Se você usa Linux, recomendo o Gedit ou o Kate.

2.5 Resumo

O que fizemos até agora é bem simples, mas já nos dá algumas informações importantes sobre o PHP. Primeiro, você deve ter percebido que um programa em PHP começa com `<?php`. É este pequeno trecho que diz **a partir daqui, o PHP deverá processar os códigos**.

Outra coisa que podemos perceber é que a instrução `echo` é usada para **imprimir** informações para o navegador.

Este arquivo também nos mostra que, assim como em outras linguagens, em PHP uma linha de instruções termina com `;` (ponto e vírgula).

Um último aprendizado deste arquivo é a função `date()`, que recebe um formato de data para formatar a data atual do computador. Usamos algumas opções de formatação como `Y` para exibir o ano com quatro dígitos e `H` para exibir a hora no formato 24 horas.

2.6 Desafios

Ao final de cada capítulo colocarei alguns desafios como forma de praticar o que foi visto até aquele momento e até forçar um pouco mais. Dessa forma, você pode pesquisar mais sobre PHP e resolver novos problemas.

Agora que já conseguimos exibir a data e a hora, tente fazer os desafios abaixo:

- Na função `date()`, experimente mudar o `Y` para `y`. O que acontece?
- Você consegue exibir a hora no formato de 12 horas, am e pm?
- E se você tivesse que exibir o dia da semana? Como faria?
- Exiba quantos dias faltam para o próximo sábado. Por exemplo, se hoje for quarta, então faltam 3 dias para sábado.
- Exiba também o nome do mês atual.

Capítulo 3

Construindo um calendário com PHP

Ok, nosso primeiro programa apenas exibia a data e a hora do processamento realizado pelo PHP. Agora vamos criar uma página que exibe um calendário.

Neste capítulo, conheceremos mais sobre o PHP. Leia com atenção, faça e até refaça os exercícios. É fundamental para compreender melhor algumas partes do PHP, então fique de olho nas novidades que serão apresentadas.

3.1 Definindo nosso calendário

Nossa página exibirá um calendário, usaremos tabelas HTML que serão montadas pelo PHP a cada nova requisição, dessa forma teremos sempre o calendário gerado pelo PHP, sem ter que criar a página manualmente.

Como estão seus conhecimentos HTML? Hora de desenferrujar e praticar, mas usaremos PHP para escrever uma parte do nosso HTML.

Ao final desde capítulo, você terá desenvolvido um programa PHP que vai gerar um calendário parecido com o abaixo:

Dom	Seg	Ter	Qua	Qui	Sex	Sáb
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31				

Figura 3.1: Exemplo de como ficará o calendário

Perceba que o calendário é apenas uma tabela HTML. Aliás, repare também que você poderá adicionar esta tabela em outros programas em PHP que você esteja desenvolvendo! Isso é bem legal, seu primeiro componente PHP.

Vamos em frente. Neste capítulo você também aprenderá novos comandos e conceitos do PHP.

3.2 Começando o calendário

Vamos começar definindo a primeira parte do calendário, crie um novo arquivo chamado `calendario.php` e salve em `c:\xampp\htdocs`. Lembre-se que um arquivo salvo nesta pasta poderá ser acessado usando o `localhost` através do navegador.

Pois bem, vamos ao código:

```
<table border="1">
    <tr>
        <th>Dom</th>
        <th>Seg</th>
        <th>Ter</th>
        <th>Qua</th>
        <th>Qui</th>
        <th>Sex</th>
        <th>Sáb</th>
```

```
    </tr>
</table>
```

Pronto, bem simples. Acesse o endereço `localhost/calendario.php` e você verá uma saída parecida com esta:

Figura 3.2: Cabeçalho do calendário

Mas, espera um pouco. Isso é HTML! Não iríamos escrever PHP? Este é mais um atributo interessante do PHP: **Podemos escrever HTML dentro de arquivos PHP**. Veja este outro exemplo de integração de HTML e PHP:

```
<h1><?php echo "Título dentro do H1"; ?></h1>
```

Perceba que o conteúdo da tag `h1` é **Título dentro do H1** e este conteúdo foi adicionado usando o `echo` do PHP. Note também que iniciamos o PHP com `<?php`, assim como da outra vez, mas desta vez também fechamos o PHP com `?>`. Após o `?>`, podemos voltar a colocar HTML.

Podemos iniciar e fechar o PHP diversas vezes dentro de uma estrutura HTML, mas devemos nomear o arquivo como `.php`. Tente fazer este exemplo, em um novo arquivo chamado `hoje.php`:

```
<html>
    <head>
        <title>Dia <?php echo date('d'); ?></title>
    </head>
    <body>
        <h1>Estamos em <?php echo date('Y'); ?></h1>
        <p>
            Agora são <?php echo date('H'); ?> horas e
            <?php echo date('i'); ?> minutos.
        </p>
    </body>
<html>
```

3.3 Usando funções

Bem, voltando ao assunto do calendário, vamos adicionar uma função ao arquivo `calendario.php` para desenhar uma nova linha na tabela. Uma linha deve conter sete colunas, para os sete dias da semana:

```
<?php
    function linha()
    {
        echo "
            <tr>
                <td></td>
                <td></td>
                <td></td>
                <td></td>
                <td></td>
                <td></td>
                <td></td>
            </tr>
        ";
    }
?>

<table>
    ...
</table>
```

Adicionamos a função antes da tabela. Agora vamos adicionar as linhas:

```
<table border="1">
    <tr>
        <th>Dom</th>
        <th>Seg</th>
        <th>Ter</th>
        <th>Qua</th>
        <th>Qui</th>
        <th>Sex</th>
        <th>Sáb</th>
    </tr>
    <?php linha(); ?>
    <?php linha(); ?>
    <?php linha(); ?>
```

```
    <?php linha(); ?>
    <?php linha(); ?>
</table>
```

Adicionamos cinco chamadas à função `linha()`, assim, quando acessarmos `localhost/calendario.php` será renderizada uma tabela parecida com esta:

Figura 3.3: Calendário ainda sem os dias

Agora vamos adicionar uma nova função para desenhar o calendário. Esta função será chamada de `calendario()` e deverá ser adicionada logo após a função linha:

```
<?php
    function linha()
    {
        ...
    }

    function calendario()
    {
        $dia = 1;
        $semana = array();
        while ($dia <= 31) {
            array_push($semana, $dia);

            if (count($semana) == 7) {
                linha($semana);
                $semana = array();
            }

            $dia++;
        }
```

```
    }
?>
```

Aqui temos bastante coisa nova. Pela primeira vez estamos usando variáveis no PHP. Repare que as variáveis sempre começam com um cifrão (`$`). Esta é uma regra do PHP: elas sempre iniciam com cifrão seguido de uma letra ou um `underline`. Sendo assim, as variáveis `$dia`, `$semana`, `$pessoa` e `$_nome` são válidas para o PHP, mas as `$1`, `$-nome` e `$!nome` são inválidas.

Neste trecho também estamos usando um tipo de dados do PHP chamado de **array**.

Reparou no uso da instrução `while`? Viu que esta instrução é praticamente igual ao `while` de linguagens como `C`? Outra instrução bem parecida é o `if`, que verifica se uma condição é verdadeira.

Para desenhar o calendário, iniciamos no dia primeiro e usamos o `while` para fazer um laço que se repetirá até o dia 31. O array `$semana` é usado para colocar os dias da semana e garantimos que ele não terá mais que sete dias usando o `if`. A função `array_push()` adiciona mais um valor em nosso array e dentro do `if` o array é reiniciado.

Ah, uma nova função apresentada foi a `count()`. Seu funcionamento é fácil de deduzir, certo? Ela conta a quantidade de itens do nosso array `$semana`.

Mas repare que a nossa função `linha()` foi chamada com um parâmetro, que é um array com os dias da semana. Então, precisamos alterar a função `linha()` para receber este array e exibir seus valores. Altere a função `linha()` para ficar como esta:

```
<?php

    function linha($semana)
    {
        echo "
            <tr>
            <td>{$semana[0]}</td>
            <td>{$semana[1]}</td>
            <td>{$semana[2]}</td>
            <td>{$semana[3]}</td>
            <td>{$semana[4]}</td>
            <td>{$semana[5]}</td>
            <td>{$semana[6]}</td>
            </tr>
```

```
            ";
    }
```

Veja que agora ela recebe um parâmetro chamado `$semana` e o conteúdo de `$semana` é exibido usando os colchetes para acessar cada um dos itens. **Atenção**: Um array inicia seus itens no número zero! Por isso, exibimos do zero ao seis.

Agora precisamos fazer mais uma alteração para exibir o calendário que temos até agora. Na tabela, retire aquelas cinco chamadas para a função `linha()` e troque por uma chamada para a função `calendario()`:

```
<table border="1">
    <tr>
        <th>Dom</th>
        <th>Seg</th>
        <th>Ter</th>
        <th>Qua</th>
        <th>Qui</th>
        <th>Sex</th>
        <th>Sáb</th>
    </tr>
    <?php calendario(); ?>
</table>
```

Agora, acesse `localhost/calendario.php`, ou atualize a página, caso já esteja aberta, e você verá um resultado parecido com este:

Dom	Seg	Ter	Qua	Qui	Sex	Sáb
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28

Figura 3.4: Calendário, mas sem todos os dias do mês

Já está parecido com um calendário! Mas ele está exibindo apenas até o dia 28! Isso está acontecendo por estarmos verificando se o número de itens na variável

`$semana` é igual a 7. O problema é que este número não vai chegar a sete, já que de 28 para 31 são apenas 3 dias de diferença.

3.4 Entendendo e se entendendo com os erros

Uma possível solução seria chamar a função `linha()` após o laço `while` passando o array `semana` com menos dias. Altere seu código para fazer esta chamada:

```
<?php

    function calendario()
    {
        $dia = 1;
        $semana = array();
        while ($dia <= 31) {
            array_push($semana, $dia);

            if (count($semana) == 7) {
                linha($semana);
                $semana = array();
            }

            $dia++;
        }
        linha($semana);
    }
?>
```

Atualize a página. Veja que os dias estão sendo exibidos, mas perceba a quantidade de erros que apareceram!

Notice: Undefined offset: 3 in **calendario.php** on line **9**

Notice: Undefined offset: 4 in **calendario.php** on line **10**

Notice: Undefined offset: 5 in **calendario.php** on line **11**

Notice: Undefined offset: 6 in **calendario.php** on line **12**

Dom	Seg	Ter	Qua	Qui	Sex	Sáb
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31				

Figura 3.5: Calendário completo, mas com alguns erros

Veja que o erro informado é `Undefined offset: 3`. Isso quer dizer que ele está tentando acessar o índice 3 no array `$semana` e não está encontrando. O mesmo acontece com os índices 4, 5 e 6, por isso esse monte de erros apareceu.

A importância das mensagens de erro

É comum que programadores novatos ignorem as mensagens de erro da linguagem. Isso é um problema, pois quando alguma falha acontece, a linguagem faz o melhor possível para indicar o que aconteceu. Quando topar com um erro no PHP, tente ler a mensagem e compreender o que houve. Em geral, ele diz a linha do erro e explica bem o problema.

3.5 Meu PHP não mostrou os erros!

Pode acontecer de você não enxergar os erros gerados, pois seu PHP pode estar configurado para não exibir os erros. Esta é uma configuração do PHP, assim como aquela do timezone que fizemos no exemplo do início do livro. Para fazer com que o PHP exiba os erros, altere no arquivo `php.ini` a linha:

```
display_errors = Off
```

Para

```
display_errors = On
```

3.6 Finalizando o calendário

Vamos mudar um pouco o script para não gerar erros e para exibir corretamente o nosso calendário. Altere apenas a função `linha()` para testar se os índices existem antes de exibi-los. Para isso, vamos usar um laço `for`, que é bem parecido com o `for` de outras linguagens. Dentro do laço, vamos usar a função `isset` que verifica se uma variável existe ou se um índice em um array foi definido. O código deverá ficar assim:

```php
<?php
    function linha($semana)
    {
        echo "<tr>";
        for ($i = 0; $i <= 6; $i++) {
            if (isset($semana[$i])) {
                echo "<td>{$semana[$i]}</td>";
            } else {
                echo "<td></td>";
            }
        }
        echo "</tr>";
    }
```

Agora, execute novamente o arquivo `calendario.php` e você terá um resultado bem parecido com este:

Dom	Seg	Ter	Qua	Qui	Sex	Sáb
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31				

Figura 3.6: Calendário final, sem erros e com todos os dias

3.7 Resumo

Neste capítulo foram introduzidas algumas características e funcionalidades interessantes do PHP, como a opção de usar HTML e PHP no mesmo arquivo, e o uso de funções para centralizar blocos que podem ser repetidos no fluxo do programa. Também foram tratados os laços **for** e **while**, além do uso da condicional **if** e da função `isset()` que verifica se uma variável foi definida.

3.8 Desafios

Pronto para alguns desafios? Sugiro copiar o calendário atual para fazer os testes e desafios propostos.

- Faça uma página que exiba a hora e a frase "Bom dia", "Boa tarde" ou "Boa noite", de acordo com a hora. Use a condicional **if** e a função `date()`.
- Faça com que o calendário exiba o dia atual em negrito, usando a função `date()`.
- Exiba os domingos em vermelho e os sábados em negrito.
- Faça o calendário começar em um dia que não seja um domingo.
- E um calendário do ano todo? Será que é muito complexo?

Capítulo 4

Entrada de dados com formulário

No capítulo anterior, foram apresentadas várias funcionalidades do PHP e também sua sintaxe para blocos e laços utilizados em diversas linguagens. Agora vamos a um novo projeto: a construção de uma lista de tarefas.

4.1 Definindo a lista de tarefas

Nosso projeto será composto por apenas algumas páginas, como a lista de tarefas e um formulário para adicioná-las e editá-las. Durante este projeto será necessário receber dados do usuário, no caso, as descrições das tarefas. É muito provável que você já tenha usado sites que pedem algum tipo de informação, ou mesmo sistemas online nos quais é necessário cadastrar vários tipos de dados. Pois é exatamente isso o que faremos — exibiremos um formulário perguntando algumas informações para o usuário e quando ele inserir as informações, iremos tratá-las.

4.2 O formulário de cadastro de tarefas

Trabalhar com web exige conhecimentos em HTML. E quando digo conhecimentos em HTML, estou dizendo realmente conhecer HTML e não ter medo de criar páginas usando apenas um editor de textos para escrever seu HTML. Se você já conhece um pouco (ou bastante) de web e usa editores nos quais você praticamente desenha a página, como o Dreamweaver e similares, eu recomendo que utilize apenas um editor de texto para reproduzir os exercícios deste livro. Assim fica mais fácil de entender o que está acontecendo e por que acontece desta ou daquela maneira.

Vamos ao formulário para o cadastro das tarefas. Crie uma pasta nova chamada `tarefas` dentro da pasta do `htdocs` do XAMPP (ou onde estiver a sua instalação do Apache). Nesta pasta, crie um arquivo chamado `tarefas.php`. Neste arquivo vamos criar o formulário para as tarefas.

Inicie o arquivo com o seguinte conteúdo básico:

```
<html>
    <head>
        <title>Gerenciador de Tarefas</title>
    </head>
    <body>
        <h1>Gerenciador de Tarefas</h1>
        <!-- Aqui irá o restante do código -->
    </body>
</html>
```

Nosso projeto irá começar simples e depois adicionaremos mais funcionalidades conforme a necessidade. Por isso, vamos começar pelo mais importante, o nome da tarefa a ser realizada. Este é o código que define o nosso formulário:

```
<form>
    <fieldset>
        <legend>Nova tarefa</legend>
        <label>
            Tarefa:
            <input type="text" name="nome" />
        </label>
        <input type="submit" value="Cadastrar" />
    </fieldset>
</form>
```

Não tenha medo de digitar o código HTML, lembre-se de treinar bastante, assim fica na sua memória muscular. Adicione o código do formulário logo após o comentário no HTML que diz que ali irá o restante do código.

Agora acesse esta nova página no endereço:

http://localhost/tarefas/tarefas.php

Seu formulário vai ficar bem parecido com este:

Figura 4.1: Formulário inicial para o cadastro de tarefas

4.3 Entrada de dados

Certo, agora temos um formulário sendo exibido, mas, como será que o PHP consegue pegar os dados que são informados nos formulários? Vamos fazer um experimento: no nome da tarefa, digite **Estudar PHP** e clique no botão `Cadastrar`. O que aconteceu? Aparentemente, nada, mas o texto sumiu da caixa de texto. E outra coisa muito interessante também aconteceu, o endereço do nosso script mudou! Sim, repare que agora existe uma interrogação no endereço, seguida por um texto que tem muito a nos dizer. Veja como ficou o endereço:

http://localhost/tarefas/tarefas.php?nome=Estudar+PHP

Dá para tirarmos algumas conclusões aqui, certo? O nome do `input` no formulário é `nome` e neste endereço temos um trecho `nome=`, ou seja, o que digitamos no campo foi **enviado** através do endereço, através da **URL** (vou chamar o endereço de **URL** daqui para frente).

Na primeira vez que chamamos a página, seu endereço não tinha o trecho com o nome da tarefa, mas quando digitamos o nome e enviamos, este trecho passou a aparecer depois do símbolo de interrogação (`?`). Ou seja, um formulário serve para **enviarmos os dados para algum lugar**.

Entrada de dados na web

Se você já programou algum software para desktop ou scripts para a linha de comando, já deve ter lidado com entrada de dados do usuário.

Neste tipo de software, em geral basta parar o processamento e pedir para o usuário informar um valor, e então seguir em frente usando o valor fornecido.

Na web, as coisas são um pouco diferentes. O PHP não pode ser parado para perguntar algo para o usuário, então os dados já devem existir para o PHP quando o script começar a rodar. Confuso? Na verdade é só uma maneira diferente de encarar a entrada de dados. Veja a nossa URL, por exemplo, ela tem o trecho `nome=Estudar+PHP`. Este trecho serve para inserir uma informação para o PHP, no caso, o nome da tarefa. Sendo assim, quando o script `tarefa.php` começar a executar, ele já terá esta informação.

Por isso, na web devemos sempre pensar na entrada de dados usando os formulários que irão enviar os dados para a **próxima** página.

Por enquanto, a nossa próxima página é a mesma página, mas veremos como mudar isso mais adiante.

Legal, já sabemos que o nome da tarefa foi enviado através da URL. Agora a questão é: Como pegar esse dado dentro do PHP?

4.4 Pegando os dados da URL

PHP nasceu para a web e se dá muito bem com ela. Por isso, em PHP é super simples pegar os dados fornecidos pelo usuário através de formulários e URLs. Em PHP existem algumas variáveis chamadas de **super globais**, isso significa que estas variáveis existem em todos os escopos de seus programas PHP. Ou seja, você pode acessá-las dentro de funções, dentro de laços, dentro de classes... Enfim, em qualquer lugar do seu código elas estarão disponíveis.

No nosso caso, vamos usar a super global **$_GET**. Lembra dos arrays que foram usados para fazer o calendário? `$_GET` também é um array e o mais interessante é que o PHP pega o que está na URL e coloca em `$_GET` para usarmos! Vamos fazer um teste exibindo o nome digitado logo após o nosso formulário. Para isso, após

o fechamento do formulário, ou seja, depois da tag </form>, adicione o seguinte código:

```
...

</form>
<?php
    if (isset($_GET['nome'])) {
        echo "Nome informado: " . $_GET['nome'];
    }
?>

...
```

As reticências são apenas ilustrativas, claro! Alias, de agora em diante vou colocar reticências várias vezes para ilustrar trechos que deverão ser incluídos dentro de códigos já existentes.

Neste trecho estamos usando a função `isset()` que verifica se uma variável existe, ou, neste caso, se o índice **nome** existe no array `$_GET`. Caso o índice exista, vamos exibi-lo usando o `echo`.

Em PHP, assim como em outras linguagens, é necessário definir uma variável antes de tentar usá-la. A função `isset()` é interessante para nos ajudar a verificar se uma variável existe, ou neste caso, um índice em um array, pois `isset()` não irá gerar um erro, dizendo que a variável não existe.

Atualize sua página, em http://localhost/tarefas/tarefas.php, mantendo a URL com o nome, ou digite no formulário novamente e envie os dados. Sua página deverá ficar assim:

Figura 4.2: Exibindo os dados usando $_GET

Legal, agora que sabemos pegar o valor que foi passado através da URL usando `$_GET`, podemos guardar este valor em um `array` que será a nossa lista de tarefas.

Para isso, troque o código que acabamos de fazer para exibir o nome informado por um que pegue o valor em `$_GET` e guarde em um array chamado `$lista_tarefas`:

```
...
</form>

<?php
    $lista_tarefas = array();

    if (isset($_GET['nome'])) {
        $lista_tarefas[] = $_GET['nome'];
    }
?>
...
```

Este código é bem simples, apenas verificamos se existe o índice **nome** dentro de `$_GET` e caso exista, criamos um novo índice em `$lista_tarefas` usando a sintaxe de colchetes vazios.

Agora é necessário exibir a lista de tarefas. Ainda no mesmo arquivo, antes da tag `</body>` vamos adicionar uma tabela HTML para listar as atividades. Dentro desta tabela usaremos o `foreach` para pegar cada uma das tarefas que está no array `$lista_tarefas` e exibir como linhas da tabela:

```
...

<table>
    <tr>
        <th>Tarefas</th>
    </tr>

    <?php foreach ($lista_tarefas as $tarefa) : ?>
        <tr>
            <td><?php echo $tarefa; ?></td>
        </tr>
    <?php endforeach; ?>

</table>
```

```
</body>
...
```

Repassando o código, foi criada uma tabela com apenas uma coluna, na qual o cabeçalho é a palavra **Tarefas** e as linhas serão os nomes das tarefas.

Aqui existe um laço novo do PHP. Já vimos o `while` — agora apresento o `foreach`. Este laço serve para passar por todos os índices de um array, atribuindo cada índice a uma variável que escolhemos, no caso, a variável `$tarefa`.

Uma outra novidade aqui é que não foram usadas as chaves para criar o bloco do `foreach`, assim como foram usadas chaves para o bloco do `while`. Na verdade, o `foreach` também pode ser usado com as chaves, sem problemas, mas esta forma, usando dois pontos para abrir o bloco e a palavra `endforeach` fica mais legível em templates HTML. Lembre-se que o nosso código com `while` estava apenas dentro de um grande código PHP, neste caso estamos colocando pequenos pedaços de PHP dentro do HTML. Sendo assim, fica mais legível fazer o uso desta sintaxe. Como exemplo, veja como fica o mesmo bloco usando as chaves:

```
<?php foreach ($lista_tarefas as $tarefa) { ?>

    <tr>
        <td><?php echo $tarefa; ?></td>
    </tr>

<?php } ?>
```

Não parece ter muita diferença em um trecho pequeno desses, mas perceba que `<?php } ?>` é uma linha bem genérica, que pode estar fechando um `if`, um `while`, um `foreach` ou mesmo uma função e outros blocos PHP, enquanto `<?php endforeach; ?>` é mais expressivo. Por isso, recomendo usar esta forma. Alias, você pode experimentar outros blocos com esta sintaxe, como o `if` e `endif`, `while` e `endwhile` e `for` e `endfor`. Mas prefira esta sintaxe para os templates HTML, deixando a sintaxe de chaves para os arquivos e blocos que contenham apenas PHP.

Bem, voltando ao assunto da lista de tarefas, ao enviar um nome de tarefa, a sua página deverá ficar assim:

Figura 4.3: Lista de tarefas com uma tarefa

Como estamos deixando tudo na mesma página, fica fácil adicionar uma nova tarefa, então vamos adicionar uma nova tarefa usando o formulário que está antes da tabela, no endereço http://localhost/tarefas/tarefas.php. Aqui eu digitei **Estudar HTML** e usei o botão cadastrar e o resultado foi este:

Figura 4.4: Lista de tarefas com uma tarefa

Apenas a tarefa nova é listada! Onde está a tarefa antiga?

Por que a primeira tarefa sumiu?

PHP trabalha principalmente com web e neste caso o que acontece a cada nova requisição que fazemos, seja pedindo uma página, seja enviando dados, é que o PHP interpreta tudo de novo e devolve apenas HTML para o navegador. Ou seja, ele não lembra do que aconteceu na última requisição feita!

A cada nova requisição o PHP processa tudo de novo e não guarda as variáveis para outros acessos. Isso é um problema para a nossa aplicação, já que precisamos

manter a lista das nossas tarefas. Mas calma, nem tudo está perdido! O PHP tem um recurso que nos auxiliará a solucionar isso.

4.5 Sessões no PHP

O recurso que nos auxiliará a manter os dados entre as requisições são as sessões. Uma sessão serve para mantermos uma variável especial que irá existir em todas as nossas requisições. Lembra da super global `$_GET`? As sessões são tão fáceis de usar quanto a `$_GET`, basta usar a super global `$_SESSION`. A grande diferença é que usamos a `$_GET` mais para ler informações e usaremos a `$_SESSION` para escrever e ler informações.

O uso da `$_SESSION` exige só um esforço adicional, que é chamar a função `session_start()` no começo do nosso programa. Para isso, vamos adicionar a função antes da abertura do HTML:

```
<?php session_start(); ?>
<html>
...
```

Depois disso, precisamos alterar o uso da lista `$lista_tarefas` para pegar os dados de `$_SESSION`, caso esses dados existam. Vamos mudar o `if` que verifica se existem dados em `$_GET` e vamos adicionar um novo `if` após a criação do array `$lista_tarefas` para preenchê-lo com os dados da `$_SESSION`, quando necessário:

```
<?php
...
    if (isset($_GET['nome'])) {
        $_SESSION['lista_tarefas'][] = $_GET['nome'];
    }

    $lista_tarefas = array();

    if (isset($_SESSION['lista_tarefas'])) {
        $lista_tarefas = $_SESSION['lista_tarefas'];
    }
...
?>
```

Agora, ao cadastrar algumas tarefas, o PHP irá manter os dados entre as requisições! Veja como a sua lista deve ficar:

Gerenciador de Tarefas

Nova tarefa

Tarefa: Cadastrar

Tarefas

Estudar HTML

Estudar PHP

Estudar Javascript

Figura 4.5: A lista de tarefas usando $_SESSION para manter os dados

Legal! Temos uma lista de tarefas já bem funcional! Claro que agora precisamos de mais funcionalidades, como editar e excluir uma tarefa, mas já conseguimos pegar vários conceitos da programação para web usando PHP.

Agora dá até para brincar um pouco com o visual da aplicação usando umas linhas de CSS. O meu ficou assim:

Figura 4.6: A lista usando um pouco de CSS

Se quiser pegar este CSS, basta baixar aqui: https://gist.github.com/InFog/6860949

Agora faça duas experiências: abra outro navegador e acesse o endereço da lista de tarefas em http://localhost/tarefas/tarefas.php, o que acontece? As tarefas aparecem? Não? Mas elas não estão na sessão?

Como funcionam as sessões?

Se colocamos os dados na sessão, por que eles não aparecem quando usamos outro navegador?

Para o PHP saber qual usuário é o dono de uma sessão, ele guarda algumas informações nos `Cookies` do navegador. Na verdade, a informação mais importante fica em um cookie chamado **PHPSESSID**, que guarda um código único de identificação da sessão daquele usuário que está acessando a aplicação PHP.

Nós também podemos usar os Cookies para guardar informações que serão mantidas entre as requisições. Para isso, basta usar uma outra super global do PHP chamada `$_COOKIE`. Esta super global também é um array, assim como a `$_SESSION`.

A outra experiência é atualizar a página após cadastrar uma tarefa. Faça isso usando F5 ou outra opção de atualização do navegador. O que acontece? A última tarefa cadastrada se repete! E continua se repetindo após as atualizações da página!

Veremos mais para frente como resolver estes dois problemas.

4.6 Resumo

Neste capítulo iniciamos o desenvolvimento de uma lista de tarefas. Ela ainda é bem simples e contém poucos dados, mas já estamos trabalhando com formulários e entrada de dados na web utilizando a super global `$_GET` do PHP e também estamos manipulando as sessões utilizado a super global `$_SESSION`.

Super globais são variáveis do PHP que estão disponíveis em qualquer ponto da aplicação. No caso das super globais `$_GET` e `$_SESSION`, os valores são guardados em **arrays**.

Importante: sempre que quiser usar sessões será necessário usar a função `session_start()`, sem isso a sessão simplesmente não funciona. Fica a dica :D

4.7 Desafios

Muito bem, é hora de alguns desafios:

- Usando os mesmos conceitos que vimos até agora, monte uma lista de contatos na qual devem ser cadastrados o nome, o telefone e o e-mail de cada contato.

Continue usando as sessões para manter os dados.

- Crie uma cópia do projeto até agora (pois vamos continuar nos próximos capítulos) e altere para usar a super global `$_COOKIE` em vez de usar as sessões. Para adicionar um Cookie use a função `setcookie()` que recebe o nome do cookie e um texto com seu valor. Para pegar um cookie já definido use a superglobal `$_COOKIE`.
- Depois de alterar a aplicação para usar cookies no lugar de sessões, tente abrir os cookies nas opções do navegador e veja se seus dados aparecem lá.

Capítulo 5

Tratamento de diferentes campos de formulários

No capítulo anterior, construímos a base para a nossa lista de tarefas. Agora vamos adicionar mais informações e funcionalidades ao projeto.

Antes de começar a adicionar mais informações e funcionalidade podemos parar para analisar um pouco nosso cenário atual e então decidir se podemos continuar com ele como está ou se podemos fazer algo para melhorar.

Esta pode ser uma etapa importante no desenvolvimento de uma aplicação, pois pode tornar o trabalho à frente mais simples ou mais complicado. Às vezes é melhor alterar a base da aplicação para poder evoluir de forma mais eficiente do que insistir em um código não muito bom. Não se preocupe se não conseguir fazer isso logo no começo, pois a experiência também ajuda bastante neste tipo de decisão.

5.1 Organizando o código em arquivos separados

O que temos até o momento é uma lista de tarefas que tem apenas o nome da tarefa e só permite adicionar novas tarefas. Tudo isso foi feito em apenas um arquivo que contém um pouco de HTML e um pouco de PHP. Podemos continuar com esta estrutura, mas com o tempo o arquivo vai crescer e pode ficar complicado de ler e entender um arquivo muito grande e dividido em mais de uma linguagem. Por isso, aqui entra uma decisão importante, vamos separar nossa aplicação em dois arquivos, um deles fará o processamento de entrada de dados e manipulação da sessão e o outro irá exibir o formulário de cadastro de tarefas e a lista das tarefas cadastradas.

Não adicionaremos código novo, vamos apenas separar o código atual em dois arquivos — um arquivo será o `tarefas.php` com este conteúdo:

```php
<?php

session_start();

if (isset($_GET['nome'])) {
    $_SESSION['lista_tarefas'][] = $_GET['nome'];
}
if (isset($_SESSION['lista_tarefas'])) {
    $lista_tarefas = $_SESSION['lista_tarefas'];
} else {
    $lista_tarefas = array();
}

include "template.php";
```

Perceba que apenas juntamos os trechos de PHP que antes estavam separados em apenas um arquivo que contém somente código PHP. Outro detalhe importante neste arquivo é o uso da instrução `include`, que serve para incluir o conteúdo de outro arquivo no fluxo atual. O legal do `include` é que ele adiciona o outro arquivo e todas as variáveis e funções do arquivo atual continuam valendo no arquivo incluído, por isso podemos, neste caso, incluir um arquivo com apenas o template do formulário de cadastro de tarefas e a lista de tarefas e ainda podemos continuar usando a variável `$lista_tarefas` que foi definida no arquivo `tarefas.php` e que contém um array com as tarefas cadastradas.

O arquivo com o template deve se chamar `template.php` pois este é o arquivo que estamos incluindo usando o `include` no arquivo `tarefas.php`. Veja como fica o arquivo `template.php`:

```
<html>
    <head>
        <meta charset="utf-8" />
        <title>Gerenciador de Tarefas</title>
        <link rel="stylesheet" href="tarefas.css" type="text/css" />
    </head>
    <body>
        <h1>Gerenciador de Tarefas</h1>
        <form>
            <fieldset>
                <legend>Nova tarefa</legend>
                <label>
                    Tarefa:
                    <input type="text" name="nome" />
                </label>
                <input type="submit" value="Cadastrar" />
            </fieldset>
        </form>
        <table>
            <tr>
                <th>Tarefas</th>
            </tr>
            <?php foreach ($lista_tarefas as $tarefa) : ?>
                <tr>
                    <td><?php echo $tarefa; ?></td>
                </tr>
            <?php endforeach; ?>
        </table>
    </body>
</html>
```

Agora temos apenas o `foreach` que é um código PHP, o restante é apenas HTML. Agora fica mais simples de para adicionar e alterar as funcionalidades, pois os arquivos estão menores e, o que é mais importante, estão com suas responsabilidades separadas.

Devo separar meus projetos em vários arquivos sempre?

Esta é uma pergunta difícil e a resposta é um pouco vaga: Depende.

Tudo depende do tamanho que sua aplicação terá e do quanto você julgar que vale a pena investir um tempo separando os as responsabilidades entre os arquivos. No geral, vale bastante a pena fazer esta separação.

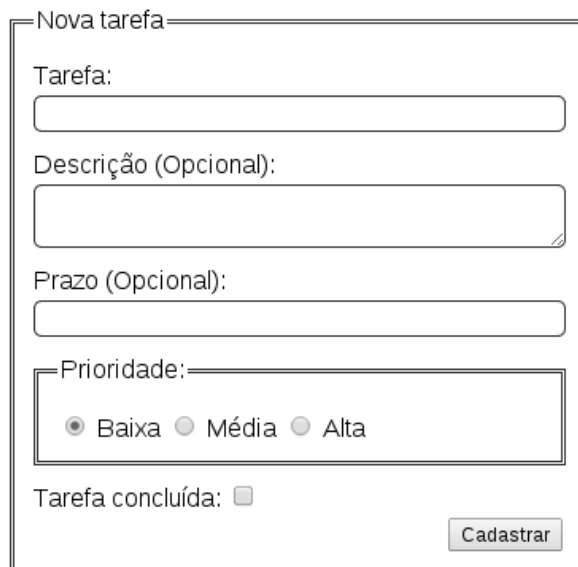
Veja que no nosso caso o programa ainda está bem pequeno e mesmo assim a separação de arquivos já melhorou bastante no entendimento das partes.

No geral não estamos perdendo tempo quando paramos para organizar melhor um projeto. Esse tempo investido agora pode se tornar uma grande economia de tempo nas futuras manutenções do código.

5.2 Adicionando mais informações às tarefas

Para que nosso sistema de controle de tarefas fique mais prático, vamos adicionar algumas informações para melhor descrever e organizar as tarefas.

Por enquanto já temos um título para a tarefa, agora vamos adicionar uma descrição, um prazo para conclusão, um nível de prioridade e uma confirmação de que a tarefa já está concluída. Para isso, vamos adicionar os novos campos no formulário HTML:

```
...

<label>
    Descrição (Opcional):
    <textarea name="descricao"></textarea>
</label>
<label>
    Prazo (Opcional):
    <input type="text" name="prazo" />
</label>
<fieldset>
    <legend>Prioridade:</legend>
    <label>
        <input type="radio" name="prioridade" value="baixa" checked />
```

```
        Baixa

        <input type="radio" name="prioridade" value="media" />
        Média

        <input type="radio" name="prioridade" value="alta" />
        Alta
    </label>
</fieldset>
<label>
    Tarefa concluída:
    <input type="checkbox" name="concluida" value="sim" />
</label>
<input type="submit" value="Cadastrar" />

...
```

Se você usar o CSS dos exemplos, seu formulário deverá ficar parecido com este:

Figura 5.1: Formulário completo para o cadastro de tarefas

O legal é que só precisamos alterar o arquivo `template.php`, enquanto o `tarefas.php` continuou o mesmo. O sistema continua funcionando e apenas ignora os dados adicionais enviados.

Agora vamos alterar também o `tarefas.php` para pegar as novas informações. Vamos usar o campo com o nome da tarefa como a base para saber se devemos cadastrar uma nova tarefa, já que o nome da tarefa é a única informação essencial para nós.

Para isso, vamos alterar aquele `if` logo após o `session_start()` e também seu conteúdo. É claro que ele vai ficar maior, pois agora são vários campos que precisamos pegar do formulário enviado:

```
<?php

session_start();

if (isset($_GET['nome']) && $_GET['nome'] != '') {
    $tarefa = array();

    $tarefa['nome'] = $_GET['nome'];

    if (isset($_GET['descricao'])) {
        $tarefa['descricao'] = $_GET['descricao'];
    } else {
        $tarefa['descricao'] = '';
    }

    if (isset($_GET['prazo'])) {
        $tarefa['prazo'] = $_GET['prazo'];
    } else {
        $tarefa['prazo'] = '';
    }

    $tarefa['prioridade'] = $_GET['prioridade'];

    if (isset($_GET['concluida'])) {
        $tarefa['concluida'] = $_GET['concluida'];
    } else {
        $tarefa['concluida'] = '';
    }
```

```
    $_SESSION['lista_tarefas'][] = $tarefa;
}
...
```

Repare que usamos sempre a função `isset()` para saber se os índices de `$_GET` existem. Isso é necessário pois o navegador não envia os campos em branco, nem os campos desmarcados.

Agora é só alterar a tabela para exibir todos os dados:

```
<table>
    <tr>
        <th>Tarefa</th>
        <th>Descricao</th>
        <th>Prazo</th>
        <th>Prioridade</th>
        <th>Concluída</th>
    </tr>
    <?php foreach ($lista_tarefas as $tarefa) : ?>
        <tr>
            <td><?php echo $tarefa['nome']; ?></td>
            <td><?php echo $tarefa['descricao']; ?></td>
            <td><?php echo $tarefa['prazo']; ?></td>
            <td><?php echo $tarefa['prioridade']; ?></td>
            <td><?php echo $tarefa['concluida']; ?></td>
        </tr>
    <?php endforeach; ?>
</table>
```

Acesse o formulário e cadastre algumas tarefas. Veja como ficou a minha página com duas tarefas cadastradas:

Gerenciador de Tarefas

Nova tarefa

Tarefa:

Descrição (Opcional):

Prazo (Opcional):

Prioridade:

Baixa Média Alta

Tarefa concluída:

Cadastrar

Tarefa	Descricao	Prazo	Prioridade	Concluída
Estudar PHP	Estudar PHP e também HTML, sessões são legais!	03/07/2013	media	
Comprar Leite	Passar na padaria e comprar leite	01/07/2013	alta	sim

Figura 5.2: Formulário e página com novos campos para o cadastro de tarefas

5.3 Conclusão do capítulo e do uso de sessões

Neste momento temos um sistema de gerenciamento de tarefas já bem interessante. Durante o desenvolvimento até aqui já foram usadas funcionalidades do PHP para manter a sessão e para incluir arquivos com trechos de código. Graças a esta separação em arquivos o nosso projeto ficou mais legível e separamos as responsabilidades de cada arquivo, um para tratar os dados e outro para exibir o formulário e a lista de tarefas.

Nesta questão de separação de arquivos poderíamos ir além, separando um arquivo para o formulário e outro para a lista de tarefas. Mas isso fica para você testar e decidir se vai ficar mais organizado ou se vai separar demais as responsabilidades.

Por enquanto, nossa aplicação permite apenas a adição das tarefas e funciona apenas enquanto o navegador está aberto, pois quando trocamos de navegador, ou quando reiniciamos o navegador atual, a sessão se perde e com isso perdemos as atividades cadastradas. Isso quer dizer que as sessões são ruins? Não! Isso quer dizer apenas que elas não servem para o nosso problema atual, que é guardar a lista de tarefas e acessar de outro navegador sem perder os dados. Mas em cenários onde é necessário, por exemplo, guardar o login do usuário, uma sessão é ideal.

5.4 Resumo

Neste capítulo adicionamos novos campos ao formulário das tarefas e tratamos estes campos através do PHP. Um ponto bem importante foi a separação do programa em mais de um arquivo, sendo que cada um é responsável por uma parte do geral, tornando-se menor e mais simples para manter.

Nos próximos capítulos vamos passar a guardar os dados em um banco de dados.

5.5 Desafios

Agora, mais alguns desafios para treinar e evoluir:

- Continue a lista de contatos que está nos desafios do capítulo anterior. Além do nome, do telefone e do e-mail de cada contato, adicione um campo para descrição, um para a data de nascimento e um checkbox para dizer se o contato é favorito ou não.
- Separe as responsabilidades da lista de contatos, mantendo um arquivo para cada, assim como fizemos na lista de afazeres.

Capítulo 6

Acessando e usando um banco de dados

Nos últimos capítulos construímos uma lista de tarefas que funciona muito bem quando usamos apenas um navegador e mantemos a sessão sempre aberta. Para a nossa aplicação este não é o cenário ideal, por isso é necessário guardar nossos dados em um local mais adequado, de onde vamos conseguir recuperá-los sempre que necessário, mesmo estando em outro navegador ou fechando o navegador e abrindo-o novamente.

É para resolver este tipo de problema que entra o banco de dados e, no nosso caso, o famoso MySQL.

Como nossa aplicação já tem uma estrutura bacana e já funciona bem usando as sessões, o que precisamos fazer é trocar o uso das sessões pelo uso do banco de dados. Por isso, vamos começar por criar o banco de dados, definindo quais dados queremos guardar e faremos com que o PHP se conecte ao banco para poder inserir e recuperar esses dados.

A manipulação do banco de dados é feita com a utilização de outra linguagem, a **SQL**, que nos permite executar comandos para inserir, pesquisar e remover dados, fazer relatórios etc. Neste livro veremos apenas um básico sobre a SQL, focando mais nos comandos necessários para o desenvolvimento dos projetos. Se você ainda não tem familiaridade com SQL, recomendo que também invista um tempo estudando esta linguagem, pois você poderá usar SQL para trabalhar com diversos bancos de dados além do MySQL, integrando com várias outras linguagens de programação além do PHP.

6.1 O banco de dados MySQL

MySQL é um software livre para banco de dados. Isso significa que você pode usar o MySQL em seus projetos e ainda pode contribuir com o desenvolvimento do próprio MySQL, se assim desejar ou necessitar.

O MySQL é bastante usado em aplicações web por sua versatilidade e por ser suportado em diversas plataformas e diferentes linguagens.

Por padrão, o MySQL não tem uma interface gráfica para administração e utilização, mas é possível encontrar diversas ferramentas que fazem este tipo de trabalho, sendo que uma das mais conhecidas é o MySQL Workbench. Não usaremos o MySQL Workbench, mas você pode pesquisar mais sobre este software e usá-lo para administrar seus bancos MySQL.

6.2 Instalando o MySQL

Como já fizemos a instalação do XAMPP, o MySQL veio junto. Então, só precisamos usar o Painel no XAMPP para iniciar o serviço. Fique sempre de olho para ver se não é necessário liberar o serviço do MySQL no firewall.

MySQL no Linux

Para quem usa Linux, mais especificamente os derivados do Debian e do Ubuntu, basta instalar o MySQL com o seguinte comando:

```
sudo apt-get install mysql-server
```

Usuários de Mac OS X que instalaram o XAMPP poderão iniciar o MySQL usando o painel do XAMPP.

6.3 PHPMyAdmin, administrando o banco de dados

A instalação do XAMPP contém uma ferramenta muito interessante para a gestão de bancos MySQL, o **PHPMyAdmin**. Esta é uma ferramenta escrita em PHP usada para gerenciar o MySQL, com opções para criar novos bancos, usuários, tabelas, inserir, pesquisar e remover registros etc.

Instalação do PHPMyAdmin

Se você optou pelo XAMPP, o PHPMyAdmin já vem instalado e pronto para usar. Caso a sua instalação não seja com o XAMPP, ou outros pacotes neste estilo, você também poderá baixar o PHPMyAdmin em seu site oficial: http://www.phpmyadmin.net/

Caso você use Ubuntu, Debian ou outras distros Linux, procure pelo pacote do PHPMyAdmin. No Debian/Ubuntu basta usar o apt para fazer a instalação:

```
apt-get install phpmyadmin
```

Se você optou pelo XAMPP, será necessário iniciar o serviço do MySQL para poder acessar o PHPMyAdmin. Para isso, acesse o painel do XAMPP e clique na opção `Start` no serviço MySQL:

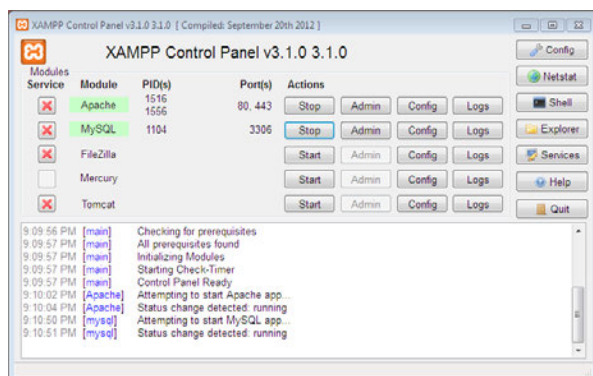



Figura 6.1: Painel do XAMPP, clique em Start no serviço MySQL

Lembre-se de que pode ser necessário autorizar o serviço no firewall do Windows. Se este for o caso, ele irá abrir uma janela pedindo autorização.

Para acessar PHPMyAdmin utilize o endereço http://localhost/phpmyadmin. Você poderá ver uma página parecida com esta:

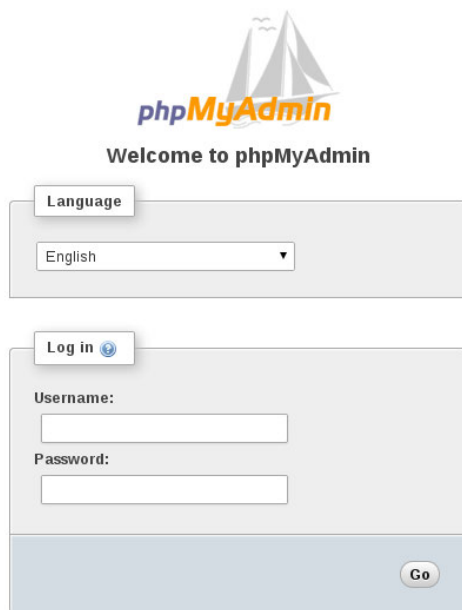



Figura 6.2: Página de login do PHPMyAdmin

Nesta página você pode escolher o idioma para usar o PHPMyAdmin. Acesse seu PHPMyAdmin usando o usuário **root** e a senha **root**.

Atenção! O XAMPP pode entrar direto na página de administração do PHPMyAdmin, sem pedir uma senha! Isso é normal em algumas versões dele.

Após acessar você verá uma página como esta:

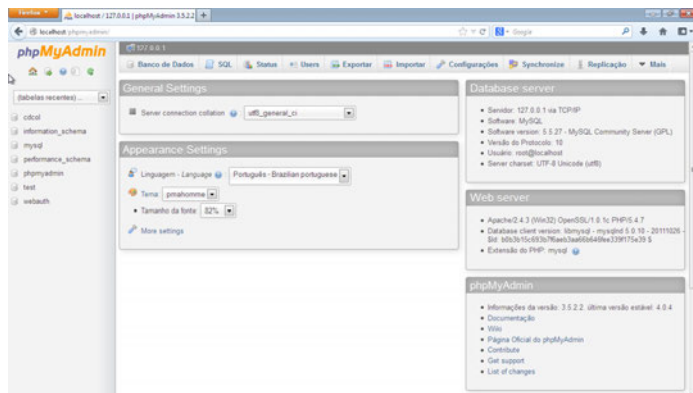



Figura 6.3: Página inicial do PHPMyAdmin

6.4 Criando o banco de dados

Vamos criar um novo banco de dados. Se você escolheu o idioma português na página de login, selecione a opção `Bancos de dados`. Na próxima página digite o nome do banco de dados, que será `tarefas`, na caixa de entrada **Criar banco de dados**. Na opção `Colação`, ou `Collation`, selecione a opção `utf8_general_ci`.

Figura 6.4: Formulário para criar um novo banco de dados no MySQL

Agora, basta clicar no botão `Criar` e o novo banco deve aparecer no menu à esquerda:

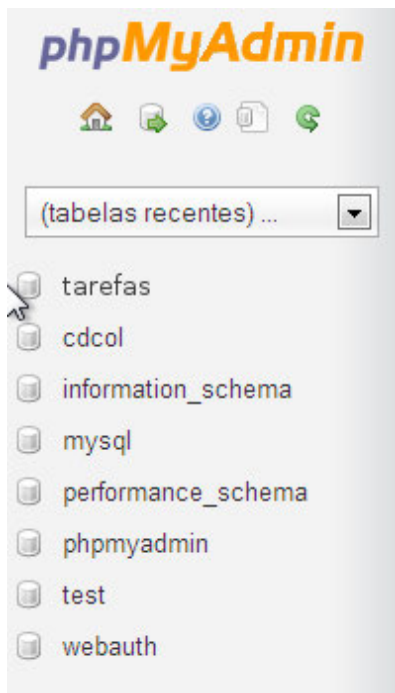



Figura 6.5: Veja o banco 'tarefas' na lista dos bancos

Clicando no nome, seremos informados de que o banco ainda não possui tabelas.

6.5 Criando a tabela

Nosso projeto é um gerenciador de tarefas, então vamos precisar apenas de uma tabela com os campos necessários para guardar os dados que já temos nos arrays de tarefas dentro da `$_SESSION`. Nossa tabela deverá ficar assim:

tarefas	
•id	integer
•nome	varchar(20)
°descricao	text
°prazo	date
•prioridade	integer(1)
•concluida	boolean

Figura 6.6: Modelagem da tabela tarefas

Repare que a tabela é muito parecida com os arrays que armazenam as tarefas, as diferenças são pequenas, como o campo `id` e os campos `nome`, que foi definido com até 20 caracteres, e `prioridade` que é um número inteiro.

O campo `id` será uma identificação única das nossas tarefas e será um número crescente. Dessa forma teremos a tarefa 1, 2, 3 e assim por diante, sem nunca repetir o número. Isso é importante para identificarmos as tarefas e não misturarmos quando precisarmos fazer referência a uma tarefa. Imagine que usássemos o campo `nome` para isso, fatalmente teríamos duas tarefas com o mesmo nome, e isso atrapalharia na hora de saber qual é qual.

O campo `nome` agora tem um limite de até 20 caracteres. Isso é algo comum em bancos de dados. Já que não precisamos de nomes muito grandes para as tarefas, 20 caracteres devem ser suficientes.

O campo `prioridade` é um número com apenas um algarismo. Vamos usar desta forma pois fica mais simples guardar no banco as prioridades 1, 2 e 3, em vez de baixa, média e alta. Isso também é bastante comum em bancos de dados, pois reduz o espaço utilizado e fica fácil de controlar na aplicação.

Existem duas maneiras de criar a tabela, a primeira é executando diretamente o código SQL a segunda é usando a interface do PHPMyAdmin. Vou mostrar a primeira opção pois exige menos passos e, acredite, é mais simples depois que se aprende um pouco de SQL.

Usando o PHPMyAdmin e estando com o banco `tarefas` selecionado, clique na aba `SQL` e você verá uma página com um campo para digitar comandos SQL. A página é parecida com esta:

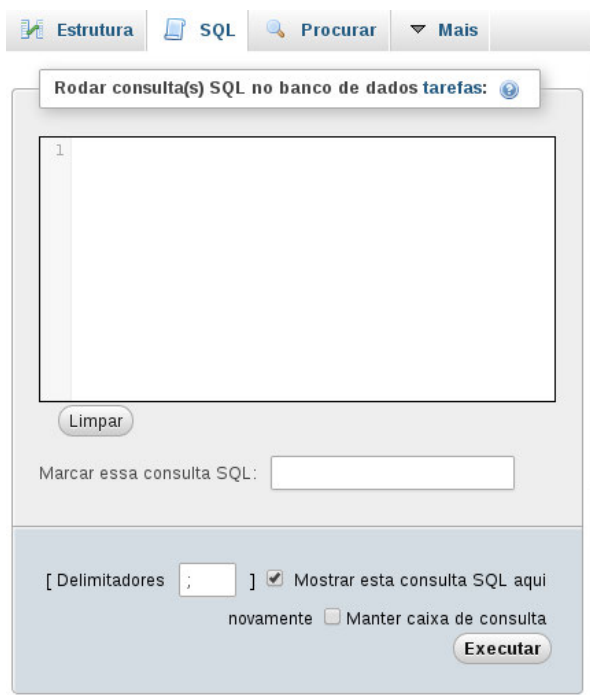



Figura 6.7: Campo para digitar comandos SQL

O código para a criação da tabela é este:

```
CREATE TABLE tarefas (
    id          INTEGER AUTO_INCREMENT  PRIMARY KEY,
    nome        VARCHAR(20) NOT NULL,
    descricao   TEXT,
    prazo       DATE,
    prioridade  INTEGER(1),
    concluida   BOOLEAN
);
```

Digite este código no campo e clique no botão `executar`. Com isso, a tabela será criada e já conseguiremos manipular seus dados.

6.6 Cadastrando e lendo os dados de uma tabela

Com a tabela pronta, podemos treinar um pouco mais de SQL antes de fazer a integração com o PHP. Vamos usar SQL para adicionar dados na tabela. O comando usado para inserir os dados em uma tabela é o `INSERT` e nele devemos informar quais são os campos que queremos preencher e seus valores.

Sempre que quisermos executar códigos SQL usando o PHPMyAdmin, basta acessarmos o banco e usar a aba `SQL`. Agora veja um exemplo para inserir os dados de uma tarefa na tabela `tarefas`:

```
INSERT INTO tarefas
(nome, descricao, prioridade)
VALUES
('Estudar PHP', 'Continuar meus estudos de PHP e MySQL', 1)
```

Após inserir os dados, use o botão `Executar`.

Repare na estrutura do comando: nós estamos dizendo para o MySQL quais os campos que queremos preencher, que não são todos os campos da tabela. Neste caso, os campos que não preenchermos, serão vazios. Atenção, é muito importante manter a mesma sequência dos campos informados e seus valores. Veja que estamos colocando a mesma sequência de **nome**, **descricao** e **prioridade** na lista de campos e também na lista de valores.

Agora, vamos selecionar os dados da tabela. O comando para selecionar os dados é o `SELECT`. Coloque o comando na caixa de texto na aba `SQL`:

```
SELECT * FROM tarefas
```

Com isso o PHPMyAdmin irá para uma nova página com o resultado da pesquisa, parecida com a próxima imagem:

	id	nome	descricao	prazo	prioridade	concluida
Editar Copiar Remover	1	Estudar PHP	Continuar meus estudos de PHP e MySQL	NULL	1	NULL

Figura 6.8: Resultado do SELECT

No comando `SELECT` nós informamos quais campos queremos, no caso, usei o **asterisco**, que significa **todos os campos**, mas também podemos indicar os campos que queremos. Faça o `SELECT` abaixo para buscar apenas dois campos da tabela:

```
SELECT nome, prioridade FROM tarefas
```

Agora o resultado será com apenas os campos `nome` e `prioridade`.

Adicione mais algumas tarefas na tabela, usando o comando `INSERT` que vimos há pouco. Eu cadastrei quatro tarefas. O meu `SELECT`, com os campos `nome`, `descricao` e `prioridade` trouxe este resultado:

←T→	nome	descricao	prioridade
Editar Copiar Remover	Estudar PHP	Continuar meus estudos de PHP e MySQL	1
Editar Copiar Remover	Estudar HTML	HTML é importante!	2
Editar Copiar Remover	Comprar leite	Desnatado	1
Editar Copiar Remover	Arrumar as fotos das	Só quando der tempo	3

Figura 6.9: Resultado do SELECT com mais linhas

Uma curiosidade interessante é que **SQL** não te obriga a escrever os comandos da linguagem, como `SELECT`, `INSERT` etc., usando caixa alta, mas é interessante manter este padrão pois simplifica a leitura. Veja que nos nossos códigos SQL até aqui existem comandos como o `FROM` e o `VALUES`, que eu não necessariamente disse para que servem, mas como eles estão em caixa alta, você conseguiu ler os comandos e deduzir o que é da linguagem e o que são nossos dados. Então, mantenha seu SQL com as palavras da linguagem em caixa alta, vai facilitar para você e para outras pessoas lerem seu código e entender o que ele faz.

6.7 Filtrando os resultados do SELECT

Usando o `SELECT` podemos filtrar os resultados da busca, por exemplo, se quisermos exibir as tarefas com a prioridade 1, faremos o SQL assim:

```
SELECT nome, descricao, prioridade FROM tarefas
WHERE prioridade = 1
```

O resultado deve ser algo similar a isso:

←T→	nome	descricao	prioridade
Editar Copiar Remover	Estudar PHP	Continuar meus estudos de PHP e MySQL	1
Editar Copiar Remover	Comprar leite	Desnatado	1

Figura 6.10: Resultado do SELECT filtrando com a prioridade 1

Também podemos filtrar por um texto que esteja dentro de outro texto, para isso usamos a instrução LIKE do SQL:

```
SELECT nome, descricao, prioridade FROM tarefas
WHERE nome LIKE '%php%'
```

O LIKE aceita o uso daqueles sinais de porcentagem. Eles servem para dizer algo como:

Procure por qualquer coisa, seguido de php, seguido de qualquer coisa

Então, se a nossa pesquisa fosse assim:

```
SELECT nome, descricao, prioridade FROM tarefas
WHERE nome LIKE 'php'
```

Ela não teria resultados. Consegue entender o motivo? Se pensar um pouco fica simples. Neste caso, ele pesquisa resultados nos quais o campo nome seja igual a php, sem textos antes ou depois. Também podemos usar a porcentagem apenas antes ou apenas depois de um texto.

No próximo capítulo vamos fazer a conexão da nossa aplicação PHP com o banco MySQL.

6.8 Resumo

Neste capítulo usamos a linguagem SQL para manipular dados em um banco de dados MySQL. O MySQL pode ser instalado à parte, ou junto com o pacote do XAMPP. Usamos também o PHPMyAdmin como ferramenta para acessar o banco através do navegador.

Para manipular o banco de dados usamos a instrução CREATE TABLE para criar uma tabela, INSERT INTO para cadastrar dados em uma tabela, SELECT para buscar os dados da tabela e também filtramos os resultados do SELECT usando WHERE e LIKE.

6.9 Desafios

Hora de alguns desafios, dessa vez voltados para bancos de dados e para a linguagem **SQL**:

- Faça mais testes com o SELECT, o WHERE e o LIKE usando o banco tarefas.

- Crie um banco chamado `contatos` para guardar os dados dos contatos do desafio iniciado nos desafios anteriores. Qual tabela será necessária para este banco? Quais campos serão necessários para cadastrar os dados que já estão sendo capturados pelo formulário? Não se esqueça de deixar um campo `id` para guardar a chave que identificará um contato como único, assim como foi feito para as tarefas.

- Crie um banco chamado `estacionamento` para cadastrar os veículos estacionados. Neste banco crie uma tabela chamada `veiculos` com os campos `id`, `placa`, `marca`, `modelo`, `hora_entrada` e `hora_saida`. Decida os tipos de dados que devem ser usados para cada campo. Cadastre alguns veículos e tente fazer pesquisas, como buscar todos os veículos de uma marca.

Capítulo 7

Integrando PHP com MySQL

No capítulo anterior, criamos o banco de dados e uma tabela para guardar os registros de nossas tarefas. Além disso, inserimos algumas tarefas no banco e usamos as buscas da linguagem SQL para selecionar as tarefas no banco e filtrar pelo nome e pela prioridade. Neste capítulo, iremos finalmente conectar o PHP ao MySQL! Como já temos a aplicação pronta, mas usando sessões, e o banco pronto, nossa tarefa será menos complexa, pois teremos que alterar apenas algumas partes da aplicação.

7.1 PHP e MySQL

PHP e MySQL já são velhos amigos. É bem comum encontrar aplicações que fazem uso destas tecnologias em conjunto. Desde pequenos sites pessoais, até grandes sistemas de gestão e lojas virtuais, a web está repleta de casos de sucesso desta parceria.

O que torna esta parceria tão forte e estável é uma combinação de fatores, como o fato de o MySQL ser um banco fácil de aprender, leve e rápido, o PHP ser fácil e flexível e ambos possuírem licenças permissivas para uso pessoal, comercial e em projetos de software livre.

PHP torna simples o uso do MySQL através de funções que conectam, executam código SQL e trazem os resultados para a aplicação. E é exatamente isso que faremos agora!

7.2 Conectando ao MySQL

Primeiro, vamos fazer a parte mais simples, que é conectar ao banco e buscar os dados que estão cadastrados nele.

O primeiro passo para realizar a conexão com o MySQL é criar um usuário no MySQL para que o PHP possa se autenticar. Este é um passo importante e ajuda a manter o banco de dados mais seguro. É como aquele usuário e senha que você precisa digitar quando liga o computador, ou quando vai acessar seu e-mail ou ainda o seu perfil em uma rede social.

Para criar o usuário no MySQL, acesse o PHPMyAdmin e selecione o banco `tarefas` e, estando nele, use a opção `Privilégios` no menu superior. Na página que abrir, clique na opção `Adicionar usuário` — uma janela abrirá pedindo os dados do novo usuário. Eu preenchi os campos conforme a imagem a seguir:

Figura 7.1: Criação de usuário no MySQL

No campo novo usuário informei **sistematarefa**, na senha digitei **sistema** e confirmei, depois cliquei no botão `Adicionar usuário`.

Após a adição do usuário, teremos um usuário chamado **sistematarefa** com a senha **sistema**. Estes são os valores que informaremos ao PHP.

Agora vamos criar um novo arquivo para fazer a conexão com o banco. Ele será gravado na mesma pasta dos arquivos da lista de tarefa, ou seja a pasta `tarefas`, e seu nome será `banco.php`:

```
<?php

$bdServidor = '127.0.0.1';
$bdUsuario = 'sistematarefa';
$bdSenha = 'sistema';
$bdBanco = 'tarefas';

$conexao = mysqli_connect($bdServidor, $bdUsuario, $bdSenha, $bdBanco);

if (mysqli_connect_errno($conexao)) {
    echo "Problemas para conectar no banco. Verifique os dados!";
    die();
}
```

No arquivo, estamos criando quatro variáveis que guardam o endereço do servidor MySQL (que é a nossa máquina), o nome de usuário, a senha de acesso e o nome do banco que queremos acessar. É legal manter estes dados em variáveis, pois se for necessário alterar, alteramos apenas a criação da variável, sem ter que procurar onde ela está sendo usada.

Repare que no código do arquivo `banco.php` existem apenas duas funções que não usamos até agora. A função `mysqli_connect()` recebe os dados de conexão com o banco e abre a conexão. Esta conexão é guardada na variável `$conexao` e vamos precisar desta variável sempre que formos interagir com o banco. A outra função é a `mysqli_connect_errno()` que pega uma conexão e verifica se houve erros de conexão. Neste caso, a função está dentro de um `if`, para verificar se houve erros ou não. Ah, tem mais uma função nova ali, a função `die()`, que faz o que o nome diz: ela mata o programa ali mesmo, sem ler o código que existe mais para frente.

Sempre ouvi falar de mysql_connect()

Talvez você já tenha visto algum texto sobre PHP e MySQL em que a função de conexão era a `mysql_connect()`. Esta função existe, assim como várias outras funções que começam com `mysql_`, mas elas serão removidas do PHP em breve. Por isso, o ideal é usar sempre a nova e melhorada versão da biblioteca MySQL do PHP, a MySQLi. No geral, basta adicionar a letra `i` nos nomes de função MySQL e um parâmetro aqui ou ali, que tudo funciona bem, mas existem algumas diferenças entre as bibliotecas. Então, a dica é: **sempre use a versão nova da biblioteca MySQL para PHP. Sempre**.

Vamos fazer uma pequena experiência com a conexão ao banco. Acesse o arquivo banco.php diretamente pelo navegador, no endereço: http://localhost/tarefas/banco.php. Você deverá ver apenas uma página em branco, o que é bom, significa que a conexão funcionou. Agora, experimente mudar o campo de senha da conexão para uma senha incorreta e acesse o arquivo novamente. Você verá a frase **Problemas para conectar no banco. Verifique os dados!**. Ou seja, nosso código de conexão está funcional e também exibe erros caso não consiga conectar.

O erro de conexão aparece sempre, mesmo com os dados corretos

Você poderá passar pelo problema do erro de conexão de vez em quando. Caso isso aconteça, verifique se o servidor do MySQL está ativo no painel de controle do XAMPP, ou em outra ferramenta que você use para gerenciar o MySQL. Uma outra forma de verificar se o banco está funcionando normalmente é acessando o PHPMyAdmin. Se ele também não conseguir conectar, o banco poderá estar mesmo desligado.

7.3 Buscando dados no banco

A próxima alteração que faremos em nosso programa é fazer com que ele busque as tarefas cadastradas no banco de dados e não mais na sessão. Para isso, abra o arquivo `tarefas.php` e ache o trecho que verifica se a sessão existe para colocar na lista de tarefas, ou cria uma lista vazia, caso a sessão não exista. O trecho é este:

```
<?php
...

if (isset($_SESSION['lista_tarefas'])) {
    $lista_tarefas = $_SESSION['lista_tarefas'];
} else {
    $lista_tarefas = array();
}

...
```

Este trecho será eliminado! No lugar dele, ficará apenas uma nova linha que chama uma função que retornará as tarefas cadastradas no banco:

```
<?php
...

$lista_tarefas = buscar_tarefas($conexao);
```

É claro que a função `buscar_tarefas()` ainda não existe, por isso iremos criá-la dentro do arquivo `banco.php`. Então, abriremos novamente o arquivo `banco.php` e no final dele vamos colocar a nova função. Mas antes de escrevê-la, precisamos planejar um pouco o seu funcionamento.

A função `buscar_tarefas()` deverá usar uma conexão com o banco para executar comandos SQL que busquem a lista de tarefas. Após buscar as tarefas, ela deverá criar um array com os dados das tarefas e então deverá retornar esses dados.

Certo, planejando assim fica mais simples alcançar nosso objetivo. Como já temos uma variável com a conexão com o MySQL, poderemos usá-la. Vamos ao código da função:

```
<?php
...

function buscar_tarefas($conexao)
{
    $sqlBusca = 'SELECT * FROM tarefas';
    $resultado = mysqli_query($conexao, $sqlBusca);

    $tarefas = array();
```

```
    while ($tarefa = mysqli_fetch_assoc($resultado)) {
        $tarefas[] = $tarefa;
    }

    return $tarefas;
}
```

Nesta função foi criada uma variável com o comando SQL que irá buscar os dados no banco, no caso o comando `SELECT * FROM tarefas`. Logo depois, a função `mysqli_query()` usa a variável com a conexão e a variável com o comando para ir ao banco, executar o comando e trazer o resultado. Este resultado fica armazenado na variável `$resultado`. Na sequência, é criada a variável `$tarefas`, que é apenas um array vazio. O próximo bloco é o `while`, que é executado até que a função `mysqli_fetch_assoc()` tenha passado por todas as linhas do resultado, sendo que cada linha do resultado é armazenada na variável `$tarefa`, no singular, para diferenciar do array `$tarefas`. E então temos, finalmente, o `return` que devolve os dados para quem chamou a função.

Agora podemos atualizar a página `tarefas.php` e ver o resultado. Acesse o endereço http://localhost/tarefas/tarefas.php e veja o que acontece. Um erro! Ele diz que a função `buscar_tarefas()` não existe! Mas nós acabamos de escrevê-la.

Analisando o problema é fácil identificar que o arquivo `tarefas.php` tem um **include** apenas no final para o arquivo `template.php`. Como ele não sabe da existência do arquivo `banco.php`, ele gera o erro. Vamos corrigir o problema adicionando um **include** para o arquivo `banco.php`. Este novo **include** será adicionado no começo do arquivo, logo após a função `session_start()`:

```
<?php
session_start();

include "banco.php";

...
```

Agora sim, atualize a página e você verá que ela exibe os dados cadastrados no banco! Veja como ficou a minha:

Gerenciador de Tarefas

Nova tarefa

Tarefa:

Descrição (Opcional):

Prazo (Opcional):

Prioridade:

Baixa Média Alta

Tarefa concluída:

Cadastrar

Tarefa	Descricao	Prazo	Prioridade	Concluída
Estudar PHP	Continuar meus estudos de PHP e MySQL		1	
Estudar HTML	HTML importante!		2	
Comprar leite	Desnatado		1	
Arrumar as fotos	Quando der tempo		3	

Figura 7.2: Listando as tarefas cadastradas no banco de dados

A importância de uma boa estrutura

Veja como conseguimos alterar a nossa lista de tarefas para pegar os dados do banco com apenas algumas alterações pontuais.

Isso foi possível pois a nossa aplicação tem uma estrutura que favorece esse tipo de alteração e até mesmo um crescimento mais saudável.

Veja a geração da lista de tarefas, por exemplo. Se estivéssemos usando a super global `$_SESSION` diretamente lá, a alteração seria mais complicada e nos custaria mais tempo, além de poder gerar mais erros. Mas, como estamos usando um array chamado `$lista_tarefas`, bastou alterar a forma como este array é gerado. Repare que nem mesmo precisamos tocar no arquivo `template.php`.

Não se preocupe se você não conseguir criar estruturas assim logo no começo do seu aprendizado. Mas lembre-se de sempre estudar e melhorar seus códigos depois de "finalizar", pois um software dificilmente vai ficar "pronto".

7.4 Cadastrando as tarefas no banco

Bem, agora que já estamos buscando as tarefas no banco, é chegada a hora de cadastrarmos mais tarefas usando o nosso formulário. Será que aqui também será possível fazer pequenas alterações para termos os dados sendo enviados para o banco, no lugar de usar sessões? Analisando o código no arquivo `tarefas.php`, conseguimos ver que a linha responsável por colocar a tarefa na sessão é esta:

```
<?php
...

$_SESSION['lista_tarefas'][] = $tarefa;

...
```

Essa linha coloca na sessão a variável `$tarefa`, que é um array com os dados da tarefa. Este array é exatamente o que precisamos para enviar nossas tarefas para o banco de dados! Assim como fizemos com o trecho que buscava as tarefas na sessão, vamos trocar esta linha por uma chamada a uma nova função que irá gravar os dados no banco. Troque a linha que acabamos de ver por esta:

```
<?php
...

gravar_tarefa($conexão, $tarefa);

...
```

A função `gravar_tarefa()` receberá a conexão que já temos, na variável `$conexao`, e também a nossa tarefa, que está no array `$tarefa`. Mas a função `gravar_tarefa()` não existe, então precisamos adicioná-la. Faremos isso no arquivo `banco.php`, pois ele está com o restante do código responsável pela comunicação com o banco MySQL. No final do arquivo iremos adicionar a função `gravar_tarefa()`:

```
<?php
...

function gravar_tarefa($conexao, $tarefa)
{
    $sqlGravar = "
```

```
        INSERT INTO tarefas
        (nome, descricao, prioridade)
        VALUES
        (
            '{$tarefa['nome']}',
            '{$tarefa['descricao']}',
            {$tarefa['prioridade']}
        )
    ";

    mysqli_query($conexao, $sqlGravar);
}
```

Veja que este código faz o que fizemos lá no PHPMyAdmin, a diferença é que ele coloca as variáveis do PHP para preencher o código SQL. Não se esqueça de colocar as variáveis PHP dentro das chaves quando estiver colocando o conteúdo das variáveis dentro de outras strings. Uma **string** é uma sequência de caracteres, ou seja, as nossas variáveis com textos são variáveis com strings. Não deixe de reparar que abrimos a string `$sqlGravar` na linha onde ela é declarada e a fechamos na linha depois do código SQL.

Nesta função também fizemos o uso da função `mysqli_query()`, pois ela é bem versátil, servindo para buscar dados e também para gravar dados. Na verdade a função `mysqli_query()` serve para **executar** código SQL, então ela devolve dados quando fizermos, por exemplo, um **SELECT**, ou apenas insere dados quando executados um **INSERT**.

Bem, a aplicação já está quase ok para funcionar usando o banco. Mas ainda é necessária uma alteração! O nosso formulário tem o campo para a definir a prioridade da tarefa e este campo está com os valores **baixa**, **media** e **alta**, mas a tabela **tarefas** no banco de dados não está preparada para estes valores, pois decidimos usar números para definir as prioridades. Sendo assim, altere o formulário para que os valores das prioridades fiquem como **1**, **2** e **3**, respectivamente, lembrando que o formulário está no arquivo `template.php`:

```
...

<legend>Prioridade:</legend>
<label>
    <input type="radio" name="prioridade" value="1" checked /> Baixa
    <input type="radio" name="prioridade" value="2" /> Média
```

```
    <input type="radio" name="prioridade" value="3" /> Alta
</label>

...
```

Muito bem, agora acesse a página em http://localhost/tarefas/tarefas.php e tente cadastrar uma tarefa. Eu cadastrei uma nova e a minha lista ficou assim:

Tarefa	Descricao	Prazo	Prioridade	Concluída
Estudar PHP	Continuar meus estudos de PHP e MySQL		1	
Estudar HTML	HTML importante!		2	
Comprar leite	Desnatado		1	
Arrumar as fotos das	Quando der tempo		3	
Estudar MySQL	Cadastrado pelo formulário		1	

Figura 7.3: Cadastrando uma nova tarefa através do formulário

Agora, precisamos exibir corretamente as prioridades na lista de tarefas, pois 1, 2 e 3 devem ser usados apenas no banco e a aplicação deve exibir sempre como **baixa**, **média** e **alta**.

A linha que precisamos alterar é a que exibe a prioridade no arquivo `template.php`:

```
...

<td><?php echo $tarefa['prioridade']; ?></td>

...
```

Adicionar alguns `ifs` podem resolver o nosso problema, mas vão deixar nosso template mais difícil de ser lido. Veja como fica se adicionarmos a logica necessária para exibir os nomes das prioridades diretamente no template:

```
<td><?php
    if ($tarefa['prioridade'] == 1) { echo 'Baixa'; }
    if ($tarefa['prioridade'] == 2) { echo 'Média'; }
    if ($tarefa['prioridade'] == 3) { echo 'Alta'; }
?></td>
```

Não está assim tão ruim, mas veja como esta outra maneira fica bem mais legível e fácil de entender:

```
<td><?php echo traduz_prioridade($tarefa['prioridade']); ?></td>
```

Bem mais simples, não? Claro que neste caso é necessário criar a função `traduz_prioridade()` e colocar nela os `ifs`, mas a grande questão aqui é que o arquivo `template.php` fica mais simples e enxuto, muito mais fácil de entender.

Pegue o arquivo `banco.php` como exemplo: nele existem as funções que criamos para lidar com o banco de dados. Graças a isso o restante da nossa aplicação precisa apenas chamar essas funções e não se preocupar em escrever SQL para executar no banco.

Vamos criar um novo arquivo chamado `ajudantes.php`. Ele irá conter funções que nos ajudarão em pequenas tarefas, como este caso de traduzir o código da prioridade para o nome dela. O conteúdo do arquivo `ajudantes.php` será apenas a função `traduz_prioridade()`, por enquanto, pois logo mais iremos colocar mais funções neste arquivo:

```
<?php

function traduz_prioridade($codigo)
{
    $prioridade = '';
    switch ($codigo) {
        case 1:
            $prioridade = 'Baixa';
            break;
        case 2:
            $prioridade = 'Média';
            break;
        case 3:
            $prioridade = 'Alta';
            break;
    }

    return $prioridade;
}
```

Repare que na função `traduz_prioridade()` utilizamos uma nova instrução do PHP, a **switch/case**. Esta instrução serve para os casos em que uma variável pode ter diversos valores e cada valor significa um fluxo diferente para o programa. Você deve estar pensando que o mesmo resultado poderia ser obtido usando alguns **ifs**, e você está certo. Mas tente escrever esta função usado apenas **if** e **else**, você verá que

ela ficará maior e mais complexa de se entender. No caso do **switch/case** as opções ficam mais claras e de fácil localização.

Mas o **switch/case** não pode ser usado em casos nos quais precisamos comparar se um valor é maior ou menor que outro. Usando **if** podemos colocar uma condição como " `$variavel < 10`", algo impossível no **case**, que só aceita valores fixos. É como se no **if** só se pudesse usar " `$variavel == 10`".

Com a função `traduz_prioridade()` pronta, precisamos dizer para a nossa aplicação que ela existe. Para isso, no arquivo `tarefas.php`, vamos adicionar mais uma instrução **include**, logo abaixo da inclusão do arquivo `banco.php`:

```
<?php
    session_start();

    include "banco.php";
    include "ajudantes.php";

    ...
```

Atualize a página e você deverá ver as prioridades com os nomes e não mais com os códigos.

Ajudantes?

É bem comum encontrar em aplicações PHP, ou de outras linguagens, arquivos, pacotes, módulos etc., com o nome de **helpers**, que significa exatamente **ajudantes**.

Em geral, estes arquivos contêm pequenos trechos de código para auxiliar em tarefas recorrentes, como no nosso caso de exibir o tipo de prioridade ou a formatação de datas, que veremos ainda neste capítulo.

Você também pode agrupar os ajudantes para determinadas tarefas em arquivos separados. Alguns exemplos seriam ajudantes para desenhar tabelas em um arquivo chamado `ajudantes_tabelas.php` ou um para criar formulários em um arquivo `ajudantes_formularios.php`. Tudo vai depender das suas necessidades e organização do código.

7.5 Cadastrando o prazo das atividades

Até o momento, estamos gravando no banco apenas o nome, a descrição e a prioridade das tarefas. Vamos agora incluir o prazo.

Este é um campo que vai dar um pouco mais de trabalho, pois no Brasil nós usamos a data no formato DIA/MÊS/ANO, mas para o MySQL o formato da data é ANO-MÊS-DIA. Por isso vamos precisar de dois tradutores: um para pegar a data que digitarmos e traduzir para o formato do MySQL e outro para pegar a data do MySQL e traduzir para exibir na lista de tarefas.

Em programação sempre existem diversas maneiras de se resolver um problema. Neste caso não é diferente, pois podemos transformar as datas usando diversas técnicas diferentes. A técnica que faremos pode não ser a melhor ou a mais elegante, mas vai resolver o nosso problema e ficará bem simples de entender.

Vamos analisar melhor o problema. Nos dois formatos de data temos os mesmos valores, mas em posições diferentes e usando separadores diferentes. Então, o que precisamos fazer é arrumar uma forma de separar os valores e reconstruir a string colocando cada valor em sua posição usando os separadores esperados, que podem ser barras ou traços.

Vamos começar pela tradução da data digitada no formulário, no formato DIA/MÊS/ANO para o formato do MySQL. A estratégia aqui será separar os valores para montar a string com o novo formato. Para isso, vamos usar uma função bem interessante do PHP, a `explode()`, que divide uma string em partes, de acordo com algum separador informado. O retorno da função `explode()` é um **array** com as partes da string, veja um exemplo:

```
<?php

$texto_original = "Frodo;Sam;Merry;Pippin";
$hobbits = explode(";", $texto_original);

foreach ($hobbits as $hobbit) {
    echo $hobbit . "<br />";
}
```

A variável `$texto_original` contém os nomes dos quatro hobbits separados por ponto e vírgula. A função `explode()` recebe dois parâmetros, sendo que o primeiro é o separador que desejamos e o segundo é o texto que desejamos separar. O resultado é um **array** no qual cada item é um dos pedaços após a separação.

No caso da tradução do prazo para as nossas tarefas, vamos criar uma função chamada `traduz_data_para_banco()` no arquivo `ajudantes.php`. Esta função irá separar os valores de dia, mês e ano e irá retornar uma string com o formato correto para o MySQL. Veja como ela fica:

```
<?php

function traduz_data_para_banco($data)
{
    if ($data == "") {
        return "";
    }

    $dados = explode("/", $data);

    $data_mysql = "{$dados[2]}-{$dados[1]}-{$dados[0]}";

    return $data_mysql;
}

...
```

Quando separamos a string da data, temos um array em que o índice zero é o dia, o índice 1 é o mês e o índice 2 é o ano. Basta construir uma nova string com os dados nas posições corretas e separando com o traço, assim temos o formato esperado pelo MySQL.

Outro detalhe é que, antes de tentar traduzir a data, verificamos se ela está vazia, já que é um campo opcional. Caso esteja vazia, retornamos uma string vazia, assim o MySQL irá gravar uma data também vazia.

Agora precisamos alterar o trecho onde montamos o array com os dados enviados para usar a nova função, lá no arquivo `tarefas.php`:

```
<?php
...

if (isset($_GET['prazo'])) {
    $tarefa['prazo'] = traduz_data_para_banco($_GET['prazo']);
} else {
    $tarefa['prazo'] = '';
}
```

```
...
```

Certo, já temos a função e também já estamos traduzindo o prazo que for digitado no formulário. Só falta alterar a função `gravar_tarefa()` no arquivo `banco.php` para também gravar o prazo. Para isso é necessário mudar a criação do código SQL. Uma nota importante é que a data precisa ser formatada como uma string para o MySQL, ou seja, ela precisa estar entre aspas. Veja como deverá ficar o código:

```
<?php
...

function gravar_tarefa($conexao, $tarefa)
{
    $sqlGravar = "
        INSERT INTO tarefas
        (nome, descricao, prioridade, prazo)
        VALUES
        (
            '{$tarefa['nome']}',
            '{$tarefa['descricao']}',
            {$tarefa['prioridade']},
            '{$tarefa['prazo']}'
        )
    ";

    mysqli_query($conexao, $sqlGravar);
}

...
```

Pronto, todas as pontas ligadas. Experimente cadastrar uma tarefa com um prazo e veja como fica. A minha lista ficou assim:

Tarefa	Descricao	Prazo	Prioridade	Concluída
Estudar PHP	Continuar meus estudos de PHP e MySQL		Baixa	
Comprar leite	Desnatado		Média	
Prova de PHP	Estudar até o prazo	2013-08-05	Baixa	

Figura 7.4: O prazo da atividade está no formato do MySQL

Quase perfeito, não? Agora precisamos traduzir a data de volta para o nosso formato brasileiro para exibir na lista de tarefas. Seguindo uma lógica bem parecida, vamos criar uma função `traduz_data_para_exibir()`, que também estará no arquivo `ajudantes.php`:

```
<?php
...

function traduz_data_para_exibir($data)
{
    if ($data == "" OR $data == "0000-00-00") {
        return "";
    }

    $dados = explode("-", $data);

    $data_exibir = "{$dados[2]}/{$dados[1]}/{$dados[0]}";

    return $data_exibir;
}

...
```

Repare nas diferenças entre a função que traduz para o banco e a que traduz para exibição. Estamos separando a string usando separadores diferentes e depois montamos novamente também com separadores diferentes. Uma outra diferença na `traduz_data_para_exibir()` é que o `if` logo no começo faz duas verificações, pois a data no banco pode estar em branco ou pode estar como uma string "0000-00-00". Para fazer esta verificação utilizamos a instrução `OR`, colocando uma condição de cada lado, assim o PHP irá verificar se uma das condições, ou as duas, são verdadeiras.

Atualize a sua lista de tarefas e veja o resultado, agora com a data correta:

Tarefa	Descricao	Prazo	Prioridade	Concluída
Estudar PHP	Continuar meus estudos de PHP e MySQL		Baixa	
Comprar leite	Desnatado		Média	
Prova de PHP	Estudar até o prazo	05/08/2013	Baixa	

Figura 7.5: O prazo da atividade com o formato correto usado no Brasil

7.6 Marcando uma tarefa como concluída

Dos campos que temos no formulário, o único que ainda não estamos gravando no banco é o campo que indica que se a tarefa está concluída. No checkbox do formulário colocamos o valor como `sim`, mas no banco precisamos do número zero para tarefas não concluídas e do número 1 para tarefas concluídas. Por isso, vamos mudar o checkbox para o valor `1`, no arquivo `template.php`:

```
<label>
    Tarefa concluída:
    <input type="checkbox" name="concluida" value="1" />
</label>
```

Agora, vamos alterar a parte onde este dado é adicionado no array `$tarefa`, no arquivo `tarefas.php`:

```
<?php
    ...

    if (isset($_GET['concluida'])) {
        $tarefa['concluida'] = 1;
    } else {
        $tarefa['concluida'] = 0;
    }

    ...
```

Repare que, após verificar se o dado está definido na super global `$_GET`, o valor está sendo definido como 1, ou como 0 quando não estiver definido. Esta verificação é assim pois quando um checkbox não é marcado, ele não é enviado pelo navegador junto com o formulário. Por isso, basta verificar se ele está ou não definido usando o `isset()`.

Agora, é necessário alterar a função `gravar_tarefa()` para gerar o código SQL incluindo o campo `concluida`. Vale lembrar que a função `gravar_tarefa()` está no arquivo `banco.php`:

```
<?php
...

function gravar_tarefa($conexao, $tarefa)
{
    $sqlGravar = "
        INSERT INTO tarefas
        (nome, descricao, prioridade, prazo, concluida)
        VALUES
        (
            '{$tarefa['nome']}',
            '{$tarefa['descricao']}',
            {$tarefa['prioridade']},
            '{$tarefa['prazo']}',
            {$tarefa['concluida']}
        )
    ";
}

...
```

Repare que o campo foi adicionado na lista de campos e o valor na lista de valores. Como o campo é um número, não precisamos usar aspas desta vez.

Cadastre uma atividade e veja o resultado:

Tarefa	Descricao	Prazo	Prioridade	Concluída
Estudar PHP	Continuar meus estudos de PHP e MySQL		Baixa	
Comprar leite	Desnatado		Média	
Prova de PHP	Estudar até o prazo	05/08/2013	Baixa	
Comprar uma mochila	Comprar uma mochila nova		Alta	1

Figura 7.6: Mostrando a tarefa como concluída, mas mostra '1' no lugar de 'Sim'

Bem, ainda não está bom para o usuário, pois agora ele vai exibir o número 1 para tarefas concluídas e o número 0 para atividades não concluídas.

Vamos adicionar mais uma função para traduzir conteúdos do banco para apresentação. No arquivo `ajudantes.php` criaremos a função `traduz_concluida()`, que retorna `Sim` ou `Não` de acordo com o campo de tarefa concluída no banco:

```
<?php
    ...

    function traduz_concluida($concluida)
    {
        if ($concluida == 1) {
            return 'Sim';
        }

        return 'Não';
    }
```

Precisamos alterar o arquivo `template.php` para usar a nova função para exibir o status de conclusão das tarefas:

```
...

<td><?php echo traduz_concluida($tarefa['concluida']); ?></td>

...
```

Atualize a página, ou cadastre uma nova tarefa e você verá a lista com as palavras **Sim** e **Não** na coluna **Concluída**:

Tarefa	Descricao	Prazo	Prioridade	Concluída
Estudar PHP	Continuar meus estudos de PHP e MySQL		Baixa	Não
Comprar leite	Desnatado		Média	Não
Prova de PHP	Estudar até o prazo	05/08/2013	Baixa	Não
Comprar uma mochila	Comprar uma mochila nova		Alta	Sim

Figura 7.7: Lista exibindo o status de conclusão como Sim e Não

Neste momento, temos uma lista de tarefas funcional, mas ainda não podemos editar as tarefas e teremos problema se digitarmos datas incorretas, nomes muito grandes ou se atualizarmos a página assim que adicionarmos uma tarefa. No próximo capítulo, vamos lidar com alguns destes problemas e vamos praticamente concluir nossa aplicação de gestão de tarefas.

7.7 Resumo

Neste capítulo alteramos a nossa aplicação para trabalhar com o banco de dados no lugar das sessões. Experimente, neste momento, fechar o seu navegador e abri-lo novamente, ou então acesse a lista de tarefas em um navegador diferente. Perceba que as atividades estão lá! Ou seja, não dependemos mais do uso das sessões para que nossa aplicação guarde os dados.

Este capítulo também reforçou o conceito da separação de responsabilidades em diferentes arquivos e também mostrou o conceito dos arquivos com funções ajudantes, que também são conhecidos no mundo da programação como **helpers**.

Também fizemos a tradução de datas no formato usado no Brasil para o formato esperado pelo MySQL e vice-versa.

7.8 Desafios

Hora de treinar fazendo mais alguns desafios:

- Continue a lista de contatos dos desafios anteriores, mas agora passando a guardar os dados em um banco de dados. Utilize todos os conceitos vistos até aqui, como a separação em arquivos, o uso de ajudantes e o uso das funções `mysqli_` do PHP para lidar com o MySQL.
- Crie a aplicação para trabalhar com o banco de dados **estacionamento**, criado nos desafios do capítulo anterior. Mais uma vez, utilize todos os conceitos que vimos até aqui.

Capítulo 8

Edição e remoção de registros

Nossa lista de tarefas funciona apenas para inclusão de tarefas e ainda não valida as informações fornecidas pelo usuário. Neste capítulo, vamos adicionar mais funcionalidades para editar e remover tarefas e com isso deixar a aplicação praticamente pronta para ser usada no dia a dia.

8.1 Edição de tarefas

Para editar nossas tarefas, precisamos de uma forma de identificarmos uma tarefa como única. Usar o nome ou o prazo não é uma boa ideia, pois tarefas podem se repetir, por isso vamos usar o campo `id` da tabela `tarefas` no banco de dados. Este campo já foi feito para esta situação, em que precisamos identificar um registro como único. Como este campo é do tipo autonumeração, cada tarefa terá seu próprio número, sem se repetir.

Vamos alterar a lista de tarefas para incluir uma nova coluna **Opções**. Esta nova coluna irá conter links para edição e remoção de tarefas. A primeira opção que adi-

cionaremos é a de edição das tarefas. Para isso, edite o arquivo `template.php`, adicionando a linha que cria a coluna **Opções** e a linha que cria o link para edição:

```
...

<tr>
    <th>Tarefa</th>
    <th>Descricao</th>
    <th>Prazo</th>
    <th>Prioridade</th>
    <th>Concluída</th>
    <th>Opções</th> <!-- A nova coluna Opções -->
</tr>
<?php foreach ($lista_tarefas as $tarefa) : ?>
    <tr>
        <td>
            <?php echo $tarefa['nome']; ?>
        </td>
        <td>
            <?php echo $tarefa['descricao']; ?>
        </td>
        <td>
            <?php echo traduz_data_para_exibir($tarefa['prazo']); ?>
        </td>
        <td>
            <?php echo traduz_prioridade($tarefa['prioridade']); ?>
        </td>
        <td>
            <?php echo traduz_concluida($tarefa['concluida']); ?>
        </td>
        <td> <!-- O campo com os links para editar e remover -->
            <a href="editar.php?id=<?php echo $tarefa['id']; ?>">
                Editar
            </a>
        </td>
    </tr>
<?php endforeach; ?>

...
```

Adicionei comentários no HTML anterior para ficar mais simples identificar quais são as novas adições. Veja a construção do link para a edição das tarefas. Ele

irá nos levar para um arquivo PHP chamado `editar.php` e está informando um parâmetro que é o **id**. Este **id** é preenchido pelo PHP pegando o id cadastrado no banco de dados.

Agora temos uma decisão importante a ser tomada: podemos criar uma nova página, com um novo formulário, para a edição das tarefas, o que vai ser mais simples; ou podemos reaproveitar o formulário que já existe, o que será um pouco menos simples — mas vai facilitar bastante as futuras alterações em nossa aplicação, pois não será necessário alterar dois formulários.

Vamos seguir o caminho do reaproveitamento e usar apenas um formulário para fazer a adição e a edição das tarefas. Para isso, será necessário alterar o nosso formulário para que ele saiba lidar com as duas situações. Mas antes, vamos fazer mais uma separação em nossa aplicação. Atualmente o arquivo `template.php` é responsável por exibir o formulário para cadastrar uma atividade e também a tabela com a lista de atividades. Isso funciona bem para a situação atual, já que não temos ainda a edição de tarefas, mas não irá funcionar bem para a edição, pois o ideal é que a edição não exiba a lista de tarefas, mas apenas o formulário.

Por isso, vamos melhorar um pouco o projeto atual separando o arquivo `template.php` em dois, um para o formulário, que irá se chamar `formulario.php` (sem acento) e outro para a tabela, que vamos chamar de `tabela.php`. Vale lembrar que, depois desta alteração, o arquivo `template.php` ainda existirá, mas com uma nova estrutura.

Este é o arquivo `formulario.php`:

```
<form>
    <fieldset>
        <legend>Nova tarefa</legend>
        <label>
            Tarefa:
            <input type="text" name="nome" />
        </label>
        <label>
            Descrição (Opcional):
            <textarea name="descricao"></textarea>
        </label>
        <label>
            Prazo (Opcional):
            <input type="text" name="prazo" />
        </label>
```

```
        <fieldset>
            <legend>Prioridade:</legend>
            <input type="radio" name="prioridade" value="1" checked />
                Baixa
            <input type="radio" name="prioridade" value="2" />
                Média
            <input type="radio" name="prioridade" value="3" />
                Alta
        </fieldset>
        <label>
            Tarefa concluída:
            <input type="checkbox" name="concluida" value="1" />
        </label>
        <input type="submit" value="Cadastrar" />
    </fieldset>
</form>
```

E o arquivo `tabela.php` vai ficar assim:

```
<table>
    <tr>
        <th>Tarefa</th>
        <th>Descricao</th>
        <th>Prazo</th>
        <th>Prioridade</th>
        <th>Concluída</th>
        <th>Opções</th>
    </tr>
    <?php foreach ($lista_tarefas as $tarefa) : ?>
        <tr>
            <td>
                <?php echo $tarefa['nome']; ?>
            </td>
            <td>
                <?php echo $tarefa['descricao']; ?>
            </td>
            <td>
                <?php echo traduz_data_para_exibir($tarefa['prazo']); ?>
            </td>
            <td>
                <?php echo traduz_prioridade($tarefa['prioridade']); ?>
            </td>
```

```
            <td>
                <?php echo traduz_concluida($tarefa['concluida']); ?>
            </td>
            <td>
                <a href="editar.php?id=<?php echo $tarefa['id']; ?>">
                    Editar
                </a>
            </td>
        </tr>
    <?php endforeach; ?>
</table>
```

O novo arquivo `template.php` ficará bem menor, já que não terá mais a lógica do formulário e da tabela. Mas agora ele terá uma nova lógica que irá verificar se a tabela deve aparecer.

Para isso vamos utilizar uma variável de controle `$exibir_tabela`, que deverá ser iniciada com o valor **false** logo no começo do arquivo `tarefas.php`. Veja primeiro como deve ficar o arquivo `template.php`:

```
<html>
    <head>
        <meta charset="utf-8" />
        <title>Gerenciador de Tarefas</title>
        <link rel="stylesheet" href="tarefas.css" type="text/css" />
    </head>
    <body>
        <h1>Gerenciador de Tarefas</h1>

        <?php include('formulario.php'); ?>

        <?php if ($exibir_tabela) : ?>
            <?php include('tabela.php'); ?>
        <?php endif; ?>
    </body>
</html>
```

Gostou desta sintaxe do **if**? Em vez de abrir chaves para os blocos, eu coloquei os dois pontos para abrir o bloco do **if** e fechei o bloco com a instrução **endif**. Esta sintaxe é bem legal para templates, pois deixa mais claro o que está sendo fechado. Lembre-se de que podemos fazer isso com o **while**, usando **endwhile**, **for** e **endfor**,

foreach e **endforeach**. Use esta sintaxe para templates — fica melhor para entender o código depois. O seu futuro eu agradecerá, e também o pessoal da sua equipe.

Agora precisamos definir a variável de controle dentro do arquivo `tarefas.php`:

```
<?php
    ...

    include "banco.php";
    include "ajudantes.php";

    $exibir_tabela = false;

    ...
```

Se você atualizar a página neste momento, verá apenas o seu título e o formulário, pois acabamos de definir como **false** a exibição da tabela.

A nossa ideia é não exibir a lista de tarefas apenas quando estivermos editando uma tarefa. Já que a nossa lógica é exibir a lista sempre e não exibir em apenas um caso, então fica mais simples se deixarmos a variável `$exibir_tabela` como **true**, em vez de **false**:

```
<?php
    ...

    include "banco.php";
    include "ajudantes.php";

    $exibir_tabela = true;

    ...
```

Não tenha medo de mudar a lógica dos seus programas de vez em quando, se for para deixar mais simples e fácil de entender. Tente evitar ao máximo a síndrome do Gollum e ficar chamando seu código de meu precioso. =)

Agora vamos cuidar daquele novo arquivo, para onde o link de edição das tarefas nos leva, o arquivo `editar.php`. Este arquivo será muito parecido com o arquivo `tarefas.php`, mas com diferenças essenciais, como a variável `$exibir_tabela` com o valor **false** e a não necessidade de usar a função `buscar_tarefas()`, já que não exibiremos as tarefas. Claro que será necessário ir ao banco de dados para pegar

os dados da tarefa que queremos editar, por isso vamos usar uma função nova, a `buscar_tarefa()`. Veja que o nome é bem parecido com a outra função, mudando apenas que a nova função busca apenas uma tarefa. É por isso que seu nome está no singular e o seu resultado será atribuído à variável `$tarefa`.

Uma outra diferença importante no arquivo `editar.php` é que ele deverá pegar também o código de identificação das nossas tarefas, que é o campo **id**.

Veja como deverá ficar o nosso arquivo `editar.php`.

```
<?php

session_start();

include "banco.php";
include "ajudantes.php";

$exibir_tabela = false;

if (isset($_GET['nome']) && $_GET['nome'] != '') {
    $tarefa = array();

    $tarefa['id'] = $_GET['id'];

    $tarefa['nome'] = $_GET['nome'];

    if (isset($_GET['descricao'])) {
        $tarefa['descricao'] = $_GET['descricao'];
    } else {
        $tarefa['descricao'] = '';
    }

    if (isset($_GET['prazo'])) {
        $tarefa['prazo'] = traduz_data_para_banco($_GET['prazo']);
    } else {
        $tarefa['prazo'] = '';
    }

    $tarefa['prioridade'] = $_GET['prioridade'];

    if (isset($_GET['concluida'])) {
        $tarefa['concluida'] = 1;
    } else {
```

```
        $tarefa['concluida'] = 0;
    }

    editar_tarefa($conexao, $tarefa);
}

$tarefa = buscar_tarefa($conexao, $_GET['id']);

include "template.php";
```

Veja que ele é praticamente uma cópia do arquivo `tarefas.php`, mas com algumas alterações fundamentais.

Precisamos criar a nova função `buscar_tarefa()`. Ela deverá ficar junto com as demais funções que lidam com banco de dados no arquivo `banco.php`:

```
<?php
...

    function buscar_tarefa($conexao, $id) {
        $sqlBusca = 'SELECT * FROM tarefas WHERE id = ' . $id;
        $resultado = mysqli_query($conexao, $sqlBusca);
        return mysqli_fetch_assoc($resultado);
    }

...
```

Repare que sempre utilizaremos o campo `id` da tabela `tarefas` para nos referenciarmos a uma tarefa como única. Isso é um padrão para os bancos de dados e, em geral, para diversas linguagens.

Agora vamos alterar o arquivo `formulario.php` para que ele já venha com os dados da tarefa preenchidos, de modo que o usuário possa fazer a edição.

Vamos primeiro adicionar o campo `id` no formulário, assim ele será enviado junto com o restante dos campos quando o usuário pedir para salvar a edição das tarefas. Este campo não precisa ser exibido para o usuário, pois não existe a necessidade de edição dele, então vamos usar um campo do tipo **hidden**:

```
<form>
    <input type="hidden" name="id"
        value="<?php echo $tarefa['id']; ?>" />
    <fieldset>
```

```
...
```

No restante dos campos, vamos preencher os valores usando o atributo `value` ou o conteúdo do elemento quando for texto, que neste caso são os campos `tarefa`, `descricao` e `prazo`. Vou colocar apenas os campos no código a seguir, mas lembre-se de que eles estão separados e existe mais código entre eles:

```
<input type="text" name="nome"
    value="<?php echo $tarefa['nome']; ?>" />

<textarea name="descricao">
    <?php echo $tarefa['descricao']; ?>
</textarea>

<input type="text" name="prazo"
    value="<?php
        echo traduz_data_para_exibir($tarefa['prazo']);
    ?>" />
```

Lembra da função `traduz_data_para_exibir()`? Olha só como ela está sendo útil mais uma vez, agora para exibir a data no formato correto para edição.

O preenchimento dos campos `prioridade` e `concluida` é um pouco diferente, já que precisamos do atributo `checked` para marcar o campo:

```
...
    <input type="radio" name="prioridade" value="1"
        <?php echo ($tarefa['prioridade'] == 1)
            ? 'checked'
            : '';
        ?> /> Baixa

    <input type="radio" name="prioridade" value="2"
        <?php echo ($tarefa['prioridade'] == 2)
            ? 'checked'
            : '';
        ?> /> Média

    <input type="radio" name="prioridade" value="3"
        <?php echo ($tarefa['prioridade'] == 3)
            ? 'checked'
```

```
            : '';
        ?> /> Alta

    <!-- Agora o campo de concluída -->

    <input type="checkbox" name="concluida" value="1"
        <?php echo ($tarefa['concluida'] == 1)
            ? 'checked'
            : '';
        ?> />
```

Não se assuste com este trecho! Aqui estamos usando o ternário do PHP, que é tipo um **if**, mas com uma sintaxe reduzida. O ternário funciona assim: você coloca uma condicional e logo depois uma opção para a condicional verdadeira e, em seguida, uma opção para a condicional falsa. No começo parece estranho, mas logo se percebe que esta forma de usar condicionais pode ser muito útil em alguns casos. Veja um exemplo com um **if** normal, igual ao que já vimos até agora:

```php
<?php

$idade = $_GET['idade'];

if ($idade >= 18) {
    echo "Bem vindo!";
} else {
    echo "Entrada proibida!";
}
```

Agora, veja o mesmo exemplo, usando ternário:

```php
<?php

$idade = $_GET['idade'];

echo ($idade > 18) ? "Bem vindo!" : "Entrada proibida!";
```

Bem mais simples, certo? Agora toda a condição cabe em apenas uma linha e fica bem fácil de ler e sem um monte de código para apenas um **echo**.

Uma pequena mudança que podemos fazer é alterar também o texto do botão `submit` no final do formulário, que atualmente é `Cadastrar`. A ideia é exibir `Atualizar` quando estivermos editando uma tarefa. Neste caso o ternário também será bastante útil:

```
...

<input type="submit" value="
    <?php echo ($tarefa['id'] > 0) ? 'Atualizar' : 'Cadastrar'; ?>
" />

...
```

Muito bem, neste momento o formulário de edição já está exibindo os dados das tarefas. Mas ainda não escrevemos a função `editar_tarefa()`. Esta função receberá um **array** com os dados da tarefa e então fará a atualização do registro no banco usando a instrução `UPDATE` da **SQL**. Veja um exemplo da instrução `UPDATE`:

```
UPDATE tabela SET
    campo1 = 'valor',
    campo2 = 123
WHERE id = 1
```

Mais uma vez vale lembrar que nossas funções que lidam com o banco de dados são apenas geradoras e executoras de SQL, por isso é importante se dedicar a aprender SQL tanto quanto ao aprendizado de PHP. ;-)

Veja como deverá ficar a função `editar_tarefa()`:

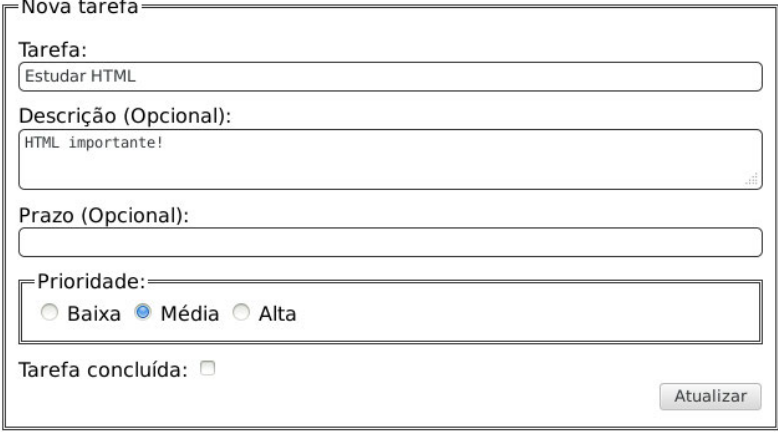
```
<?php
...

function editar_tarefa($conexao, $tarefa)
{
    $sql = "
        UPDATE tarefas SET
            nome = '{$tarefa['nome']}',
            descricao = '{$tarefa['descricao']}',
            prioridade = {$tarefa['prioridade']},
            prazo = '{$tarefa['prazo']}',
            concluida = {$tarefa['concluida']}
        WHERE id = {$tarefa['id']}
    ";

    mysqli_query($conexao, $sqlEditar);
}

...
```

Com esta função pronta, já podemos editar as tarefas. Mas se você acessar o arquivo `tarefas.php` neste momento, você verá diversos erros na página, mais ou menos conforme a imagem a seguir:

Figura 8.1: Diversos erros no formulário

Nossa! Quantos erros! Mas se analisarmos melhor veremos que na verdade todos os erros são referentes à mesma variável `$tarefa`. Este erro está acontecendo porque mudamos o formulário para preencher os dados de uma tarefa sendo editada. Já que tomamos a decisão de usar apenas um arquivo para o formulário, agora teremos que adaptar o formulário para o cadastro de novas tarefas, que antes já funcionava corretamente.

Neste caso, a correção do problema é simples: basta fornecermos uma tarefa em branco para o formulário de cadastro de tarefas, assim ele sempre terá uma variável `$tarefa` para exibir, evitando os erros de variável não definida. No arquivo `tarefas.php` adicionaremos esta nova `$tarefa` em branco, que será um array. Vamos adicionar esta variável logo antes do `include` do `template.php`:

```
<?php
...

$tarefa = array(
    'id'          => 0,
    'nome'        => '',
```

```
    'descricao'  => '',
    'prazo'      => '',
    'prioridade' => 1,
    'concluida'  => ''
);

include "template.php";
...
```

Como estamos fornecendo uma tarefa em branco, o formulário não gera mais erros. Então, vamos editar uma tarefa! Acesse o endereço http://localhost/tarefas/tarefas.php e clique no link `Editar`. Você deverá ver apenas o formulário preenchido com os dados da tarefa, mas sem a lista de tarefas embaixo:

Figura 8.2: Nosso formulário para edição de tarefas

Agora podemos editar nossas tarefas, uma vez que já temos o formulário. O arquivo `editar.php` já trata os dados e também já estamos enviando a tarefa para a função `editar_tarefa()` que usa o comando `UPDATE` do MySQL para atualizar os dados.

Mas algo ainda não está legal. Perceba que, após salvar uma tarefa, continuamos na página de edição. Esta é uma boa hora para apresentar uma nova opção: os **redirecionamentos**. A ideia aqui é quebrar o fluxo da aplicação logo após a execução da

função `editar_tarefa()`, para que o usuário seja encaminhado novamente para a página inicial — ou seja, a lista de tarefas, lá no arquivo `tarefas.php`. O PHP em si não possui uma ferramenta de redirecionamento, pois isso é feito pelo HTTP, através de informações para o navegador. Aqui eu vou mostrar uma forma simples de fazer este redirecionamento, mas lembre-se de mais tarde parar para estudar um pouco mais sobre o protocolo HTTP, ok?

Para fazer o redirecionamento, precisamos falar para o navegador que ele deve ir para um local diferente do local atual. Isso é feito através de cabeçalhos HTTP, neste caso o cabeçalho que precisamos é o `Location`. O PHP usa a função `header()` para mandar esses cabeçalhos para o navegador. Veja como fica o uso da função `header()` para redirecionarmos nossos usuários de volta para `tarefas.php` após a atualização de uma tarefa com a função `editar_tarefa()`:

```
<?php
...

editar_tarefa($conexao, $tarefa);
header('Location: tarefas.php');
die();

...
```

Logo após a função `header()`, foi adicionada a função `die()`, que encerra o processo do PHP imediatamente. A chamada da função `die()` vai evitar que o restante do arquivo `editar.php` seja executado de forma desnecessária, já que o que precisamos após a edição de uma tarefa é apenas o redirecionamento para a lista de tarefas.

Com isso, temos a lista de tarefas já funcional para cadastro e também para edição de tarefas! Mas agora precisamos de uns ajustes importantes, como a possibilidade de remover tarefas e também a validação do formulário, para evitarmos, por exemplo, o cadastro de tarefas em branco.

8.2 Remoção de tarefas

A remoção das tarefas será bem mais simples que a edição, pois dessa vez não precisamos pegar dados no banco, preencher formulário etc. Agora tudo o que precisamos é do número da tarefa para remover, ou seja, seu **id** no banco de dados.

Vamos começar adicionando o link `Remover` na nossa lista de tarefas. Este link ficará junto do link `Editar`, na coluna `Opções` da tabela, então vamos editar o arquivo `tabela.php` para fazer a adição do novo link:

```
...

<td>
    <a href="editar.php?id=<?php echo $tarefa['id']; ?>">
        Editar
    </a>
    <a href="remover.php?id=<?php echo $tarefa['id']; ?>">
        Remover
    </a>
</td>

...
```

Perceba que o link de remoção é muito parecido com o link de edição que já conhecemos. Afinal, para remover uma tarefa também vamos precisar do seu **id** no banco de dados, mas o endereço aponta para o arquivo `remover.php`, que ainda não existe.

O nosso arquivo `remover.php` também deverá utilizar o arquivo `banco.php` para acessar o banco e também deverá redirecionar o usuário após a remoção da tarefa, assim como fizemos na edição. Este arquivo será bem menor que os demais, porque precisa fazer menos trabalho do que a lista e a edição:

```
<?php

include "banco.php";

remover_tarefa($conexao, $_GET['id']);

header('Location: tarefas.php');
```

E pronto, este é o arquivo `remover.php` em todo o seu esplendor. Sim, apenas isso! É claro que ainda precisamos criar a função `remover_tarefa()` lá no arquivo `banco.php`, mas a lógica da remoção é basicamente esta.

No arquivo `banco.php`, vamos adicionar a função `remover_tarefa()`. Esta função irá utilizar o comando `DELETE` da linguagem **SQL**, que tem a sintaxe conforme este exemplo:

```
DELETE FROM tarefas WHERE id = 123
```

Este exemplo removeria o registro da tabela `tarefas` com o **id** 123. Mas também podemos fazer algo assim com SQL:

```
DELETE FROM tarefas WHERE concluida = 1
```

Este comando removeria da tabela `tarefas` todos os registros onde o campo `concluida` fosse igual a 1, ou seja, nossas tarefas concluídas.

Mas vamos voltar à função `remover_tarefa()` no arquivo `banco.php`:

```
<?php
...

function remover_tarefa($conexao, $id)
{
    $sqlRemover = "DELETE FROM tarefas WHERE id = {$id}";

    mysqli_query($conexao, $sqlRemover);
}
```

Com a função `remover_tarefa()` pronta, já podemos usar o novo link `Remover` na lista de tarefas para removermos tarefas. Faça um teste, deu certo? Se sim, vamos em frente! Caso tenha algum problema, revise seu código até aqui e leia as mensagens de erro que o PHP informa, assim fica mais fácil de resolver problemas.

8.3 Evitando o problema com a atualização de página

Ainda temos um pequeno problema com a nossa lista de tarefas! Não acredita? Então faça o seguinte: adicione uma tarefa, adicionou? Agora atualize a página usando o `F5` ou o botão de atualizar do seu navegador. Viu o que acontece? Ele repete o último cadastro feito! Veja como ficou a minha lista:

Teste	Um teste	Baixa	Não	Editar Remover
Teste	Um teste	Baixa	Não	Editar Remover
Teste	Um teste	Baixa	Não	Editar Remover
Teste	Um teste	Baixa	Não	Editar Remover

Figura 8.3: Tarefas repetidas após atualizar a página depois de uma gravação

Esse problema acontece pois, quando atualizamos a página, o navegador manda os dados novamente para o PHP. Como o PHP recebeu dados, ele acaba disparando toda a rotina de gravação de novas tarefas.

Para evitarmos este problema, vamos usar o truque que aprendemos com as páginas de edição e de remoção e vamos redirecionar o usuário de volta para a lista de tarefas após a gravação de uma tarefa. Para isso, vamos usar a função `header` logo após a função `gravar_tarefa` no arquivo `tarefas.php`:

```
<?php
...

gravar_tarefa($conexao, $tarefa);
header('Location: tarefas.php');
die();

...
```

A função `die()` aqui tem o mesmo propósito do arquivo `editar.php`, para encerrar o PHP logo após enviar o cabeçalho de redirecionamento para o navegador.

Faça um teste agora e após a gravação você verá o formulário e a lista de tarefas normalmente, e ainda poderá atualizar a página sem o problema de duplicação da última tarefa cadastrada.

8.4 Resumo

Até este momento já vimos bastante coisa. Saímos do zero e já temos um ambiente PHP e MySQL com a nossa primeira aplicação: um gerenciador de tarefas!

Neste capítulo fizemos a edição das tarefas e também sua remoção. Além disso, corrigimos alguns comportamentos usando o `Redirect`, através da função

`header()` do PHP.

Neste capítulo também foram utilizados os comandos `UPDATE` e `DELETE` da linguagem **SQL**.

No próximo capítulo faremos a validação dos dados enviados pelo usuário, assim evitamos problemas como datas inválidas.

8.5 Desafios

Existem diversas formas de melhorar o que já fizemos até aqui. Experimente desenvolver funcionalidades como:

- Um botão de `Cancelar` quando estiver editando uma tarefa.
- Um botão para duplicar uma tarefa.
- Otimizar o código para ter menos repetição. Veja que os arquivos `tarefas.php` e `editar.php` são muito parecidos. Você consegue pensar em maneiras de reutilizar partes deles? Que tal um arquivo novo com as bases para eles?
- Que tal uma opção para apagar todas as tarefas concluídas?
- Você consegue pensar em uma maneira de usar apenas um arquivo para exibir as tarefas, editar, salvar uma nova, deletar etc.? Assim não seria necessário termos os arquivos `editar.php` e `remover.php`.

Além destes desafios para a lista de tarefas, continue os outros desafios dos capítulos anteriores:

- Adicione uma opção para editar e outra para remover os contatos da lista de contatos.
- Adicione uma opção para listar apenas os contatos favoritos.
- Adicione opções para remover os dados dos veículos cadastrados no banco de dados do estacionamento.
- Faça uma opção no desafio do estacionamento para mostrar os veículos que entraram no estacionamento em uma determinada data.

Capítulo 9

Validação de formulários

Um problema comum em praticamente qualquer software que tenha que lidar com dados fornecidos por um usuário é a validação. Sabe quando você preenche um formulário em um site (ou um software desktop, ou um app para smartphone etc.) e, logo após enviar os dados, o formulário é exibido novamente com seus dados e mais alguns alertas de campos que são obrigatórios ou que foram preenchidos incorretamente? Pois então, este é um cenário onde aconteceu algum tipo de validação de entrada de dados.

9.1 Validação na lista de tarefas

Nossa lista de tarefas tem um campo obrigatório, que é o nome da tarefa e um campo que pode gerar erros se preenchido incorretamente, que é o campo de prazo, onde usamos uma data.

Atualmente estamos usando o campo de nome da tarefa para decidir se vamos gravar a tarefa ou não. Isso é um pouco ruim, pois se alguém preencher os dados

da tarefa, menos o nome, e enviar, irá perder o que foi digitado e a tarefa não será gravada.

Podemos começar validando essa entrada, para saber se o formulário foi enviado independentemente do campo de nome da tarefa ser preenchido ou não.

Mas, antes de fazer esta validação, vamos pegar um pedaço deste capítulo para falar da entrada de dados.

9.2 Entrada de dados usando POST

Até este momento estamos usando a super global `$_GET` do PHP para pegar os dados enviados pelo usuário no formulário e também nas páginas de edição e de remoção para pegar o **id** da tarefa que queremos editar ou remover.

Você já deve ter usado diversos formulários em sites e outras aplicações online e percebido que os dados digitados em um formulário não ficam expostos no endereço, assim como nosso formulário de tarefas está fazendo. Existem basicamente duas formas de enviar dados usando um formulário, uma delas é usando o GET e a outra é usando o POST.

Usando o POST, os dados do formulário não ficam mais no endereço da página de destino, o que pode tornar a aplicação mais segura. Imagine um formulário que envia uma senha e ela fica exposta na URL de destino. Nada bom, né?

Certo, mas como podemos trocar o uso do GET nos formulários da nossa lista de tarefas para o POST? O primeiro passo é alterar o formulário para que use o método POST e para isso é necessário adicionar o atributo **method** com o valor **POST** no elemento **form**. Estas alterações devem ser feitas no arquivo `formulario.php`:

```
<form method="POST">
    <input type="hidden" name="id" value="<?php echo $tarefa['id']; ?>"/>

    ...
```

Com isso, o formulário passa a enviar os dados usando o POST. O padrão dos navegadores é enviar usando GET, por isso esse método era usado quando não tínhamos informado explicitamente qual usar.

Agora também é necessário alterar toda a parte que pega os dados do formulário para cadastrar no banco, pois estamos usando a super global `$_GET`. Esta alteração será bem simples, já que o PHP cuida da entrada via POST também. Então, tudo o

que temos que fazer é trocar `$_GET` por `$_POST` nos campos relacionados ao formulário. Essas alterações devem ser feitas nos arquivo `tarefas.php` e `editar.php` Veja como fica a alteração no arquivo `tarefas.php`:

```php
<?php
    ...

    if (isset($_POST['nome']) && $_POST['nome'] != '') {
        $tarefa = array();

        $tarefa['nome'] = $_POST['nome'];

        if (isset($_POST['descricao'])) {
            $tarefa['descricao'] = $_POST['descricao'];
        } else {
            $tarefa['descricao'] = '';
        }

        if (isset($_POST['prazo'])) {
            $tarefa['prazo'] = traduz_data_para_banco($_POST['prazo']);
        } else {
            $tarefa['prazo'] = '';
        }

        $tarefa['prioridade'] = $_POST['prioridade'];

        if (isset($_POST['concluida'])) {
            $tarefa['concluida'] = 1;
        } else {
            $tarefa['concluida'] = 0;
        }

        gravar_tarefa($conexao, $tarefa);

        ...
```

As alterações no arquivo `editar.php` também são simples:

```php
<?php
    ...

    if (isset($_POST['nome']) && $_POST['nome'] != '') {
```

```
$tarefa = array();

$tarefa['id'] = $_POST['id'];

$tarefa['nome'] = $_POST['nome'];

if (isset($_POST['descricao'])) {
    $tarefa['descricao'] = $_POST['descricao'];
} else {
    $tarefa['descricao'] = '';
}

if (isset($_POST['prazo'])) {
    $tarefa['prazo'] = traduz_data_para_banco($_POST['prazo']);
} else {
    $tarefa['prazo'] = '';
}

$tarefa['prioridade'] = $_POST['prioridade'];

if (isset($_POST['concluida'])) {
    $tarefa['concluida'] = 1;
} else {
    $tarefa['concluida'] = 0;
}

editar_tarefa($conexao, $tarefa);

...
```

O uso do POST também é importante para formulários que enviam arquivos, mas veremos isso um pouco mais para frente.

9.3 Validando o nome da tarefa

Vamos à validação do nosso primeiro campo, que será o campo com o nome da tarefa a ser cadastrada. Hoje já existe uma espécie de validação que verifica se o nome da tarefa foi digitado ou não e então decide se irá cadastrar a tarefa no banco. Esse tipo de decisão vai sempre depender de o campo com o nome da tarefa ter sido digitado pelo usuário. Isso pode trazer problemas, pois o usuário pode ter digitado o restante

dos campos, mas ter se esquecido do nome da tarefa e neste caso ele vai perder tudo o que digitou no restante dos campos.

Vamos tratar primeiro da validação na página de cadastro, ou seja, no arquivo `tarefas.php`. Vamos começar alterando a linha que verifica se tem que entrar ou não no processo de gravação de novas tarefas, pois ela verifica apenas se o índice **nome** existe em `$_POST` e não necessariamente se existem mais coisas no `$_POST`. Esta é a linha:

```
<?php
    ...

    if (isset($_POST['nome']) && $_POST['nome'] != '') {

    ...
```

Esta linha será substituída por esta:

```
<?php
    ...

    if (tem_post()) {

    ...
```

A função `tem_post()` ainda não existe, então vamos adicioná-la no arquivo `ajudantes.php`:

```
<?php
    ...

    function tem_post()
    {
        if (count($_POST) > 0) {
            return true;
        }

        return false;
    }
```

Veja que a função `tem_post()` faz uma contagem do número de índices existentes em `$_POST` e retorna **true** quando este número for maior que zero, que é uma situação que irá acontecer apenas se alguma informação for enviada via **POST**.

Certo, agora o processo de gravação inicia se tiver dados no **POST**, independente de o campo **nome** ter sido enviado. Isso é bom, pois vamos validar este campo e repreencher os dados no formulário, caso a validação não passe.

Vamos criar uma nova variável de controle chamada `$tem_erros` e também um array para guardar os erros em cada campo chamado `$erros_validacao`, ambos logo acima do **if** que verifica se tem **POST**:

```
<?php
    ...

    $exibir_tabela = true;

    $tem_erros = false;
    $erros_validacao = array();

    if (tem_post()) {

    ...
```

No caso, `$tem_erros` já começa como **false**, e será alterada para **true** conforme as validações acontecerem. O array `$erros_validacao` também receberá seus valores conforme as validações acontecerem.

Para validar o campo com o nome da tarefa precisamos apenas verificar se ele não está vazio. Para isso vamos alterar a linha que pega o nome da tarefa no arquivo `tarefas.php`:

```
<?php
    ...

    $tarefa['nome'] = $_POST['nome'];

    ...
```

Esta linha está logo após o **if** que acabamos de alterar para usar a função `tem_post()`. No lugar desta linha, faremos uma verificação parecida com a verificação que já existe para os demais campos, com a diferença de que agora precisamos preencher o array com os erros e alterar a variável `$tem_erros` para **true** caso o nome da tarefa esteja vazio:

```
<?php
    ...
```

```
    if (isset($_POST['nome']) && strlen($_POST['nome']) > 0) {
        $tarefa['nome'] = $_POST['nome'];
    } else {
        $tem_erros = true;
        $erros_validacao['nome'] = 'O nome da tarefa é obrigatório!';
    }

    ...
```

A maior novidade neste trecho é o uso da função `strlen()` que conta o tamanho de um texto. O que estamos fazendo é verificar se o nome está no **POST** e se a contagem de caracteres é maior que zero. Repare que o **else** está criando um índice **nome** no array `$erros_validacao` com uma frase indicando o erro que aconteceu. Além disso, também aproveitamos o mesmo **else** para alterar a variável `$tem_erros` para **true**.

Agora que temos uma variável que diz se algum erro aconteceu, podemos usá-la para decidir se vamos ou não fazer a gravação da tarefa. Como já temos uma função que faz a gravação, podemos colocar a chamada desta função dentro de um **if**:

```
<?php
    ...

    if (! $tem_erros) {
        gravar_tarefa($conexao, $tarefa);
        header('Location: tarefas.php');
        die();
    }

    ...
```

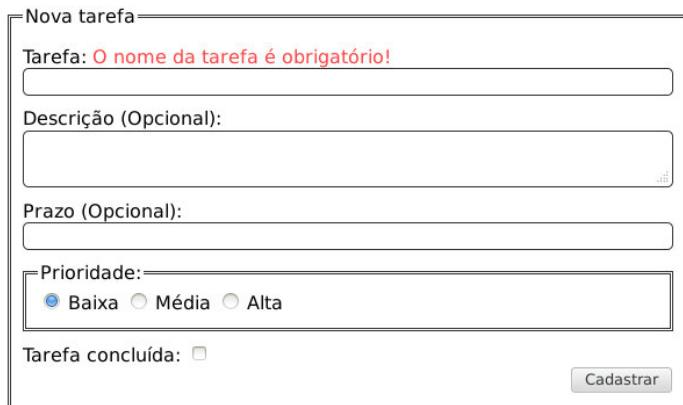
Veja que usamos a exclamação para negar o valor de `$tem_erros`, então esta condição pode ser lida como "se não tem erros". Caso algum erro exista, a função função `gravar_tarefa()` não será chamada e o arquivo `tarefas.php` continuará a ser executado, já que o usuário não será redirecionado para a lista de tarefas. Se você tentar incluir uma tarefa vazia neste momento, você simplesmente voltará ao formulário, mas sem avisos de erro.

9.4 Adicionando o aviso de erro

Para adicionar o aviso de erro vamos alterar o arquivo `formulario.php`, mais especificamente, vamos alterar a parte que cria o campo para o nome da tarefa:

```
...

<label>
    Tarefa:
    <input type="text" name="nome"
        value="<?php echo $tarefa['nome']; ?>" />
</label>

...
```

As variáveis `$tem_erros` e `$erros_validacao` serão usadas para decidir se o erro será exibido e para pegar o valor erro. Altere o trecho dentro da tag `label` para ficar assim:

```
...

<label>
    Tarefa:
    <?php if ($tem_erros && isset($erros_validacao['nome'])) : ?>
        <span class="erro">
            <?php echo $erros_validacao['nome']; ?>
        </span>
    <?php endif; ?>
    <input type="text" name="nome"
        value="<?php echo $tarefa['nome']; ?>" />
</label>

...
```

Estamos usando duas condições dentro do **if**, uma para verificar se `$tem_erros` é verdadeira e outra para verificar se o índice **nome** existe dentro do array `$erros_validacao`. Esta verificação do índice é necessária pois logo mais faremos a validação dos outros campos, então pode ser que, por exemplo, `$tem_erros` seja **true** mas que o erro não seja no campo **nome**.

O erro foi exibido dentro de um elemento **span** com a classe **erro**, assim podemos usar CSS para estilizar a frase de erro. No meu caso, eu deixei a frase em vermelho. Se você usar o CSS dos exemplos, terá o mesmo resultado.

Faça uma nova tentativa de cadastro sem digitar o nome da tarefa e você deverá ver o formulário novamente, com o erro sendo exibido:

Figura 9.1: Exibição do erro no nome da tarefa

Mas ainda temos um problema. Se experimentarmos cadastrar uma tarefa apenas com uma descrição, voltaremos para a página do formulário, mas sem os dados da descrição. Este é um ponto importante da validação, pois quando validamos um formulário e informamos para o usuário que existe algum campo faltando, não podemos apagar o restante das informações já fornecidas. Quem nunca passou por um formulário que apagou os dados por conta de apenas um campo esquecido? É bem chato, não?

Para preencher novamente os campos que já foram enviados vamos fazer uso do array `$tarefa` que é gerado antes de incluirmos o arquivo `template.php`, lá no final do arquivo `tarefas.php`:

```
<?php
    ...

    $tarefa = array(
        'id'          => 0,
        'nome'        => '',
        'descricao'   => '',
        'prazo'       => '',
```

```
        'prioridade' => 1,
        'concluida'  => ''
    );

    ...
```

Este array é gerado com dados em branco, então precisamos verificar se existem dados enviados através do **POST** para que sejam usados no lugar dos valorem em branco. Lembra do ternário? Aqui ele será bem útil:

```
<?php
...

$tarefa = array(
    'id'         => 0,
    'nome'       => (isset($_POST['nome'])) ?
        $_POST['nome'] : '',

    'descricao'  => (isset($_POST['descricao'])) ?
        $_POST['descricao'] : '',

    'prazo'      => (isset($_POST['prazo'])) ?
        traduz_data_para_banco($_POST['prazo']) : '',

    'prioridade' => (isset($_POST['prioridade'])) ?
        $_POST['prioridade'] : 1,

    'concluida'  => (isset($_POST['concluida'])) ?
        $_POST['concluida'] : ''
);

...
```

Deixei o ternário em duas linhas em cada um dos campos para facilitar a leitura aqui no livro, mas ele poderia ficar em apenas uma linha no código.

Perceba que em cada um dos campos é verificado se existe um valor no **POST** e então este valor é usado. Caso não exista, usamos um valor padrão em branco.

O campo que foge um pouco dos demais é o **prazo**. Nele estamos usando a função `traduz_data_para_banco()`. Mas, espera um pouco, na verdade não estamos enviando esta data para o banco! Então, por que usar esta função?

Vamos analisar a criação do campo do prazo, lá no arquivo `formulario.php`:

```
<label>
    Prazo (Opcional):
    <input type="text" name="prazo"
       value="<?php echo traduz_data_para_exibir($tarefa['prazo']); ?>"
    />
</label>
```

Veja que na criação do campo no formulário estamos usando sempre a função `traduz_data_para_exibir()`, que espera um texto vazio, ou uma data no formato ANO-MÊS-DIA, mas a data que entramos no formulário está no formato DIA/MÊS/ANO. Para evitar erros na formatação da data, vamos montar o array com o formato esperado pelo nosso formulário.

Agora, preencha os campos do formulário, menos o nome da atividade e tente fazer a gravação. Você verá novamente o formulário, com o erro de validação do nome, assim como da outra vez, mas agora os demais campos já estarão preenchidos com os dados informados:

Figura 9.2: Exibição de erro no nome, mas com o formulário preenchido

Se preenchermos também o nome da tarefa, ela será gravada normalmente. O próximo passo é validar o prazo, pois os demais campos são opcionais e não precisam de validação de dados corretos.

9.5 Validando a data digitada

O prazo é um campo opcional, mas existe o problema de o usuário entrar datas inválidas, como um 31 de fevereiro ou mesmo textos que não estejam no formato DIA/MÊS/ANO que estamos esperando.

Vamos focar na validação do formato primeiro, depois iremos para a validação de datas que existem.

Vamos definir então qual será o formato de data aceita em nossa aplicação. O dia poderá ter um ou dois dígitos e o mês também, mas o ano deverá ter sempre quatro dígitos. Veja alguns exemplos de datas que serão consideradas válidas e outras inválidas também:

```
# Datas válidas
03/07/2012
3/07/2012
03/7/2012
3/7/2012

# Datas inválidas
003/7/2012
3/007/12
3/7/12
03/07/12
03/07
```

Certo, tendo definido os formatos que serão aceitos pela aplicação podemos fazer a validação.

Agora, como faremos isso? Já sabemos que podemos explodir uma string usando um separador e transformá-la em um array. Aí poderíamos contar os caracteres de cada item do array para saber se a data está no formato esperado. Isso com certeza vai dar um trabalhão e não será muito prático. Veja mais ou menos como ficaria a validação desta forma:

```php
<?php
    function validar_data($data)
    {
        $partes = explode('/', $data);

        if (count($partes) != 3) {
            return false;
```

```
        }

        $dia = $partes[0];
        $mes = $partes[1];
        $ano = $partes[2];

        if (strlen($dia) < 1 OR strlen($dia) > 2) {
            return false;
        }

        if (strlen($mes) < 1 OR strlen($mes) > 2) {
            return false;
        }

        if (strlen($ano) != 4) {
            return false;
        }

        return true;
    }
```

Nada prático, né? Imagine validar o formato de um CPF assim, ou um CEP, ou um CNPJ, ou um código de barras de um boleto bancário... Enfim, deve existir algo que verifica certos padrões em um texto e diz se esse texto está no padrão.

9.6 Expressões regulares

Este é um tópico muito interessante que merece um livro só para ele, mas vamos com calma, vou mostrar apenas o básico necessário para validarmos o prazo das nossas atividades e deixar a sugestão para que você se aprofunde no assunto, pois realmente vale a pena.

Expressões regulares nos permitem pesquisar textos dentro de outros textos e verificar se determinados textos seguem um padrão predefinido. Usando expressões regulares podemos fazer coisas como a validação do formato que queremos para a nossa data do prazo de uma forma muito mais simples do que a forma manual que vimos anteriormente.

Usar expressões regulares para validar o formato da nossa data é algo mais ou menos assim:

Verifique se o texto começa com um ou dois dígitos, seguido de uma barra, seguido

por mais um ou dois dígitos, seguido por mais uma barra e por último uma sequência de quatro dígitos.

Este formato que deve ser encontrado é chamado de **padrão** e um padrão que valida, por exemplo, apenas um número é este:

```
[0-9]
```

Isso significa qualquer número de 0 até 9, ou seja, todos os dígitos do sistema decimal. Então para validar dois números seguidos poderíamos usar um padrão como o seguinte:

```
[0-9][0-9]
```

Basta repetir o padrão, assim teríamos uma validação para dois números seguidos. Mas imagine que precisássemos validar 10 números seguidos, escrever o padrão 10 vezes não é prático e com certeza iria dificultar muito o entendimento do código. Por isso, existem também quantificadores para as expressões regulares:

```
[0-9]{10}
```

Isso quer dizer 10 vezes um número. Mas, além de poder colocar uma quantidade fixa de repetições do padrão, podemos colocar intervalos:

```
[0-9]{3,5}
```

Isso significa um padrão de três a cinco números. Com isso podemos voltar ao padrão para validar o prazo das atividades:

```
[0-9]{1,2}\/[0-9]{1,2}\/[0-9]{4}
```

Ler esta expressão regular é simples: "um ou dois números, barra, um ou dois números, barra, quatro números". Perceba que existem barras invertidas antes das barras, isso é necessário para "escapar" as barras, ou seja, estamos dizendo para a expressão regular que ele não deve tratar a barra como um caractere especial.

Agora podemos melhorar um pouco a expressão, pois temos que verificar se este é o único texto digitado no campo do prazo. Por isso, precisamos adicionar os caracteres que significam o começo e o fim do texto em um padrão, que são respectivamente o acento circunflexo e o cifrão:

```
^[0-9]{1,2}\/[0-9]{1,2}\/[0-9]{4}$
```

Ufa, expressão regular já montada e pronta para usar. Aliás, uma dica legal aqui é usar o site **regexpal** para testar suas expressões regulares: http://regexpal.com/. Este foi o site que eu usei para construir e testar a nossa expressão regular:

Figura 9.3: Regexpal sendo usado para testar a validação da data

Certo, já sabemos um basicão de expressões regulares, agora precisamos aplicar a validação dentro do PHP. Para isso vamos usar a função `preg_match()`, que recebe um padrão e um texto para verificar se o texto casa com o padrão. Vamos criar uma nova função `validar_data()` no arquivo `ajudantes.php`:

```
<?php
    ...

    function validar_data($data)
    {
        $padrao = '/^[0-9]{1,2}\/[0-9]{1,2}\/[0-9]{4}$/';
        $resultado = preg_match($padrao, $data);

        return $resultado;
    }

    ...
```

Ei! Este padrão está diferente do que fizemos! Calma! Ela tem apenas uma barra no começo e outra no final, pois este é um padrão usado em algumas linguagens, inclusive na linguagem Perl, que é de onde o PHP herda algumas coisas.

Agora que já temos a função para validar o prazo, vamos adicioná-la no arquivo `tarefas.php`, já aproveitando para colocar o erro de validação:

```
<?php
    ...
```

```
    if (isset($_POST['prazo']) && strlen($_POST['prazo']) > 0) {
        if (validar_data($_POST['prazo'])) {
            $tarefa['prazo'] = traduz_data_para_banco($_POST['prazo']);
        } else {
            $tem_erros = true;
            $erros_validacao['prazo'] = 'O prazo não é uma data válida!';
        }
    } else {
        $tarefa['prazo'] = '';
    }

    ...
```

Veja que é necessário primeiro verificar se existe algum texto no campo prazo, para depois usar a função `validar_data()`, pois este é um campo opcional.

Precisamos alterar também o arquivo `formulario.php` para exibir o erro de validação do prazo:

```
<?php
    ...

    <label>
        Prazo (Opcional):
        <?php if ($tem_erros && isset($erros_validacao['prazo'])) : ?>
            <span class="erro">
                <?php echo $erros_validacao['prazo']; ?>
            </span>
        <?php endif; ?>
        <input type="text" name="prazo" value=
            "<?php echo traduz_data_para_exibir($tarefa['prazo']); ?>"
        />
    </label>

    ...
```

Agora já podemos tentar cadastrar uma tarefa com o prazo incorreto, como por exemplo "12/12/12":

Gerenciador de Tarefas

Nova tarefa

Tarefa: O nome da tarefa é obrigatório!

Descrição (Opcional):

Prazo (Opcional): O prazo não é uma data válida!

12/12/12

Prioridade:

Baixa Média Alta

Tarefa concluída:

Cadastrar

Figura 9.4: Erro de validação do prazo

Mas se digitarmos algo como "amanhã" no prazo, ou uma data como "12/12", seremos agraciados com vários erros, pois as funções `traduz_data_para_exibir()` e `traduz_data_para_banco()`, usadas para montar o array `&tarefa` e para exibir o formulário esperam datas com três partes separadas por traços ou barras. Por isso, vamos adicionar uma pequena modificação nestas duas funções para que retornem a data da forma como a receberam, caso ela não tenha três partes separadas por barras. Vale lembrar que as funções estão no arquivo `ajudantes.php`:

```
<?php
    ...

    function traduz_data_para_banco($data)
    {
        if ($data == "") {
            return "";
        }

        $dados = explode("/", $data);
```

```
        if (count($dados) != 3) {
            return $data;
        }

        $data_mysql = "{$dados[2]}-{$dados[1]}-{$dados[0]}";

        return $data_mysql;
    }

    function traduz_data_para_exibir($data)
    {
        if ($data == "" OR $data == "0000-00-00") {
            return "";
        }

        $dados = explode("-", $data);

        if (count($dados) != 3) {
            return $data;
        }

        $data_exibir = "{$dados[2]}/{$dados[1]}/{$dados[0]}";

        return $data_exibir;
    }

    ...
```

Veja que apenas adicionei um **if** para verificar se a quantidade de partes da data, após o uso da função `explode()`, é igual a três.

Agora sim, podemos usar a lista de tarefas com a validação da data e sem os erros em caso de datas em formatos incorretos.

Mas... Bem, ainda temos um pequeno problema, a validação do prazo apenas verifica se o formato de entrada da data está correto, sem considerar se a data existe ou não. Desse modo, podemos digitar datas como "99/99/9999" que não teremos um erro de validação, mesmo que esta data não exista.

Validar se uma data é válida é uma atividade bem chatinha. Sério, além de o fato de cada mês ter uma certa quantidade de dias, ainda temos os maravilhosos anos bissextos, nos quais fevereiro tem 29 dias.

Ainda bem que PHP possui a função `checkdate()` que verifica se uma data é válida. Então, vamos usá-la dentro da nossa função `validar_data()`:

```
<?php
    ...

    function validar_data($data)
    {
        $padrao = '/^[0-9]{1,2}\/[0-9]{1,2}\/[0-9]{4}$/';
        $resultado = preg_match($padrao, $data);

        if (! $resultado) {
            return false;
        }

        $dados = explode('/', $data);

        $dia = $dados[0];
        $mes = $dados[1];
        $ano = $dados[2];

        $resultado = checkdate($mes, $dia, $ano);

        return $resultado;
    }

    ...
```

A função `checkdate()` recebe três parâmetros, sendo o primeiro o mês, o segundo o dia e o terceiro o ano. Esta função retorna **true** apenas quando os dados informados formarem uma data válida. Experimente cadastrar uma atividade para o dia 30 de fevereiro, ou para 31 de abril, ou para 32 de dezembro e você sempre receberá o erro de validação dizendo que a data é inválida.

Neste momento temos a validação funcional para a adição de novas tarefas!

9.7 Validando o formulário de edição de tarefas

Com o cadastro de tarefas já funcional e validando, podemos agora alterar a edição de tarefas para que a validação também funcione. No arquivo `editar.php` adicione

a variável de controle `$tem_erros` e `$erros_validacao` logo após a variável `$exibir_tabela`:

```
<?php
    ...

    $exibir_tabela = false;
    $tem_erros = false;
    $erros_validacao = array();

    ...
```

Altere também o **if** que verifica se existe os dados foram enviados via post, que atualmente usa o campo com o nome da tarefa, pela nossa função `tem_post()`:

```
<?php
    ...

    $exibir_tabela = false;
    $tem_erros = false;
    $erros_validacao = array();

    if (tem_post()) {

    ...
```

O próximo passo é alterar o trecho que pega o nome da tarefa para que faça a validação e preencha o array de erros se necessário:

```
<?php
    ...

    if (isset($_POST['nome']) && strlen($_POST['nome']) > 0) {
        $tarefa['nome'] = $_POST['nome'];
    } else {
        $tem_erros = true;
        $erros_validacao['nome'] = 'O nome da tarefa é obrigatório!';
    }

    ...
```

E também devemos alterar a definição do campo prazo, validando com a função `validar_data()`:

```
<?php
    ...

    if (isset($_POST['prazo']) && strlen($_POST['prazo']) > 0) {
        if (validar_data($_POST['prazo'])) {
            $tarefa['prazo'] = traduz_data_para_banco($_POST['prazo']);
        } else {
            $tem_erros = true;
            $erros_validacao['prazo'] = 'O prazo não é uma data válida!';
        }
    }

    ...
```

E no final do bloco do **if** precisamos alterar a chamada da função `editar_tarefa()` para usar a variável `$tem_erros`:

```
<?php
    ...

    if (! $tem_erros) {
        editar_tarefa($conexao, $tarefa);
        header('Location: tarefas.php');
        die();
    }

    ...
```

Este trecho todo fica mesmo muito parecido com o arquivo `tarefas.php`, com apenas alguns pontos diferentes, mas importantes.

Se você tentar editar uma tarefa neste momento, já verá a validação funcionando, mas o repreenchimento dos campos não estará funcionando corretamente. Os valores exibidos em caso de erro serão exatamente os mesmos que estão cadastrados no banco, ou seja, a tarefa atual.

Vamos acertar este problema verificando se existem dados no **POST** logo após a chamada da função `buscar_tarefa()` que está bem no final do arquivo `editar.php`. Os dados que esta função retorna são armazenados na variável `$tarefa`, que é um array com os dados da tarefa. Então, o que precisamos fazer é substituir os valores recuperados do banco pelos que estão no **POST**. Vai ficar até parecido com a parte da validação, com a diferença de que não será necessário validar:

```
<?php
    ...

    $tarefa = buscar_tarefa($conexao, $_GET['id']);

    $tarefa['nome'] = (isset($_POST['nome'])) ?
        $_POST['nome'] : $tarefa['nome'];

    $tarefa['descricao'] = (isset($_POST['descricao'])) ?
        $_POST['descricao'] : $tarefa['descricao'];

    $tarefa['prazo'] = (isset($_POST['prazo'])) ?
        $_POST['prazo'] : $tarefa['prazo'];

    $tarefa['prioridade'] = (isset($_POST['prioridade'])) ?
        $_POST['prioridade'] : $tarefa['prioridade'];

    $tarefa['concluida'] = (isset($_POST['concluida'])) ?
        $_POST['concluida'] : $tarefa['concluida'];

    ...
```

Olha o nosso amigo ternário dando as caras novamente. Veja que nesta parte o que fazemos é a verificação do campo no **POST** e, quando este não é encontrado, usamos o valor padrão que já está no array `$tarefa`.

Não precisamos mais de alterações para que a validação funcione também na edição das tarefas, pois o formulário já está preparado para exibir os erros quando necessário. Então, já temos a adição e a edição de atividades funcional e com validação.

9.8 Resumo

Ufa, este foi um capítulo e tanto heim? Para fazer a validação de apenas dois campos passamos por alguns tópicos bem interessantes, como as famosas expressões regulares e também a validação de datas. Não foi necessário alterar profundamente nossa aplicação, somente alguns pontos chave para a lógica e também a exibição dos erros no formulário.

Para o repreenchimento dos campos em caso de erros de validação usamos os dados do **POST**, aproveitando a requisição atual. Existem técnicas de validação que

guardam os dados na sessão e redirecionam o usuário para uma página de erros. Você pode experimentar outras técnicas e até mesmo bibliotecas PHP especializadas em formulários e validação, mas saber o funcionamento fundamental é importante, por isso recomendo treinar mais validações desta forma que fizemos.

9.9 Desafios

Pensou que iria ficar sem desafios? Aqui estão mais alguns para praticar:

- Faça um formulário que peça um CEP e valide o valor digitado usando o padrão XXXXX-XXX, ou seja, cinco números, um traço e mais três números.
- Faça um formulário que peça um CPF e valide se o valor foi digitado no formato correto: XXX.XXX.XXX-XX (X são os números). Use expressões regulares aqui.
- Adicione validação para a lista de contatos que já está sendo feita nos desafios há alguns capítulos. Tente validar a entrada do nome, que deve ser obrigatório, do telefone, usando o formato "(00)0000-0000", do e-mail e também da data de nascimento.
- Adicione validação também para o projeto do estacionamento. As placas dos veículos devem seguir o padrão "AAA-0000", ou seja, três letras e quatro números.

Capítulo 10

Upload de arquivos

Uma tarefa bem comum em aplicações web é o upload de arquivos. Vez ou outra nós acabamos enviando fotos, documentos e outros tipos de arquivos para sites e outras aplicações web. O processo de envio de arquivos é bem parecido com o envio de dados em formulários, com a diferença de que escolhemos os arquivos em nossos computadores em vez de digitar ou selecionar os dados em formulários.

Neste capítulo, veremos como enviar arquivos para que sejam tratados pelo PHP e armazenados no sistema de arquivos, com referências cadastradas no banco de dados.

10.1 Anexos para a lista de tarefas

Nossa lista de tarefas agora também terá anexos. Imagine que o usuário queira cadastrar uma tarefa como **imprimir o relatório** e queira, é claro, anexar o relatório à tarefa.

O armazenamento de arquivos pode ser feito de algumas maneiras diferentes e

utilizaremos uma técnica que mistura o uso do banco de dados com o sistema de arquivos.

Antes de colocar a mão na massa precisamos definir como será esse sistema de anexos:

- Teremos uma página com os detalhes da tarefa, onde estará o formulário para adicionar os anexos.
- Uma tarefa poderá ter um ou mais anexos.
- Permitiremos anexar apenas arquivos `zip` ou `pdf`.
- O usuário poderá remover o anexo, sem remover a tarefa.

Uau, temos um pouco de trabalho pela frente, mas o resultado será bem interessante. Vamos começar alterando o banco de dados, depois vamos adicionar a página com os detalhes das tarefas e o formulário para adicionar anexos e, por último, faremos o tratamento do envio dos arquivos e listagem dos arquivos na página de detalhes da tarefa.

10.2 Mudanças no banco de dados

Nosso banco de dados atual possui apenas a tabela `tarefas`, então precisamos fazer alterações para que seja possível guardar os dados relacionados aos anexos. Se adicionarmos um novo campo chamado, por exemplo, `anexo` à tabela `tarefas`, poderemos adicionar apenas um anexo por tarefa, mas definimos que uma tarefa poderá ter vários anexos, então esta solução não nos atenderá.

Sendo assim, teremos que adicionar mais uma tabela em nosso banco para guardar exclusivamente os anexos. Esta nova tabela deverá guardar uma referência às tarefas para que se possa associá-las a seus anexos. Veja como fica o diagrama para a nova tabela `anexos` e sua associação com a tabela `tarefas`:

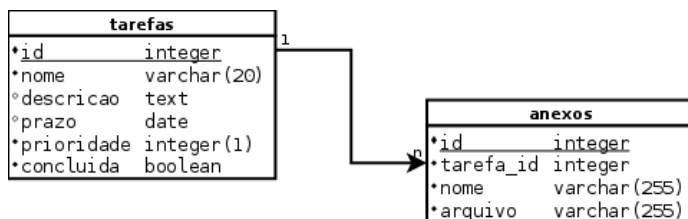



Figura 10.1: Diagrama do banco com a tabela anexos

A tabela `anexos` terá seu próprio **id** para identificação dos anexos como únicos e também um campo `tarefa_id` que irá guardar uma referência ao campo `id` da tabela `tarefas`.

Os campos `id` em cada uma das tabelas são chamados de **chaves primárias**, pois reconhecem cada registro como único. Já o campo `tarefa_id` na tabela `anexos` guarda uma referência a um registro único em outra tabela, por isso ele é chamado de **chave estrangeira**.

Além disso a nova tabela também terá dois campos para guardarmos o nome do arquivo sem sua extensão e a sua localização no sistema de arquivos.

Esse processo de separação de tabelas e uso de chaves primárias e chaves estrangeiras faz parte da **normalização de bancos de dados**. Eu recomendo que você leia mais a respeito para evitar problemas comuns de duplicação de dados e até mesmo perda de informações relacionais em bancos de dados.

Para criar a nova tabela acesse o PHPMyAdmin e siga um processo parecido com o que usamos para criar a tabela das tarefas, mas dessa vez o código SQL necessário para criar a tabela `anexos` é o seguinte:

```
CREATE TABLE anexos (
    id          INTEGER AUTO_INCREMENT  PRIMARY KEY,
    tarefa_id   INTEGER NOT NULL,
    nome        VARCHAR(255) NOT NULL,
    arquivo     VARCHAR(255) NOT NULL
);
```

Após executar este código o PHPMyAdmin deverá exibir as duas tabelas no banco `tarefas`:

Figura 10.2: As tabelas no banco tarefas

10.3 Página com os detalhes das tarefas

Com o banco de dados já pronto para receber os anexos das tarefas, vamos para a próxima etapa, que é a criação de uma página que irá exibir os dados de uma tarefa e um formulário para cadastrar anexos.

A nossa nova página ficará no endereço http://localhost/tarefas/tarefa.php?id=N, onde **N** será o **id** da tarefa que queremos exibir. Este endereço estará em um link que iremos adicionar agora na lista de tarefas, usando o nome da tarefa. No arquivo `tabela.php` altere a linha que exibe o nome, adicionando o link:

```
<td>
    <a href="tarefa.php?id=<?php echo $tarefa['id']; ?>">
        <?php echo $tarefa['nome']; ?>
    </a>
</td>
```

Este link aponta para o arquivo `tarefa.php`, que ainda não existe. Nosso próximo passo é a criação desde arquivo, que terá uma estrutura semelhante ao `tarefas.php`, mas deverá mostrar apenas uma tarefa por vez:

```
<?php

include "banco.php";
include "ajudantes.php";

$tem_erros = false;
$erros_validacao = array();

if (tem_post()) {
    // upload dos anexos
```

```
}

$tarefa = buscar_tarefa($conexao, $_GET['id']);

include "template_tarefa.php";
```

Perceba que já temos todas as funções que estão sendo usadas neste arquivo. Por enquanto, vamos deixar a parte do upload apenas com um comentário e vamos criar o arquivo `template_tarefa.php`, que está sendo incluído na última linha:

```
<html>
    <head>
        <meta charset="utf-8" />
        <title>Gerenciador de Tarefas</title>
        <link rel="stylesheet" href="tarefas.css" type="text/css" />
    </head>
    <body>
        <h1>Tarefa: <?php echo $tarefa['nome']; ?></h1>
        <p>
            <a href="tarefas.php">Voltar para a lista de tarefas</a>.
        </p>

        <p>
            <strong>Concluída:</strong>
            <?php echo traduz_concluida($tarefa['concluida']); ?>
        </p>
        <p>
            <strong>Descrição:</strong>
            <?php echo nl2br($tarefa['descricao']); ?>
        </p>
        <p>
            <strong>Prazo:</strong>
            <?php echo traduz_data_para_exibir($tarefa['prazo']); ?>
        </p>
        <p>
            <strong>Prioridade:</strong>
            <?php echo traduz_prioridade($tarefa['prioridade']); ?>
        </p>

        <h2>Anexos</h2>
        <!-- lista de anexos -->
```

```
        <!-- formulário para um novo anexo -->
    </body>
</html>
```

Este arquivo serve para exibir os dados da tarefa, a lista de anexos de cada uma e também o formulário para adição de novos anexos. Deixei os trechos com a lista de anexos e com o formulário de anexos em branco, pois vamos preencher estes trechos mais para frente. Acesse a lista de tarefas e clique no link no nome da tarefa para acessar a página com os seus detalhes. Você deverá ver uma página parecida com esta:

Tarefa: Estudar HTML

Voltar para a lista de tarefas.

Concluída: Não

Descrição: HTML é muito importante!

Prazo:

Prioridade: Média

Anexos

Novo anexo

Figura 10.3: Página com os detalhes da tarefa

10.4 O formulário para cadastrar anexos

Ainda no arquivo `template_tarefa.php` vamos criar o formulário para o envio de anexos. Este formulário será bem simples e terá apenas um campo para a seleção do arquivo, um campo com o **id** da tarefa e o botão de envio:

```
...

<!-- formulário para um novo anexo -->
<form action="" method="post" enctype="multipart/form-data">
```

```
    <fieldset>
        <legend>Novo anexo</legend>

        <input type="hidden"
            name="tarefa_id" value="<?php echo $tarefa['id']; ?>" />

        <label>
            <?php if ($tem_erros && isset($erros_validacao['anexo'])):?>
                <span class="erro">
                    <?php echo $erros_validacao['anexo']; ?>
                </span>
            <?php endif; ?>

            <input type="file" name="anexo" />
        </label>

        <input type="submit" value="Cadastrar" />
    </fieldset>
</form>

...
```

Como já temos a experiência do formulário de cadastro de tarefas, já podemos deixar o formulário de cadastro de anexos pronto para exibir erros de validação, mesmo sem ter feito a validação ainda.

Agora, repare nas diferenças do formulário de cadastro de anexos para o formulário de cadastro de tarefas. A tag `<form>` do formulário de cadastro de anexos tem a propriedade `enctype="multipart/form-data"`, que serve para indicar para o navegador que nosso formulário irá fazer o envio de arquivos. Outra diferença é o `input` do anexo, que tem a propriedade `type="file`, que serve para indicar que este campo deverá ser preenchido com um arquivo escolhido no computador do usuário. Veja como fica este formulário na página com os detalhes da tarefa:

Figura 10.4: Form para envio de anexos

10.5 Recebendo arquivos pelo PHP

Após adicionar o formulário, vamos alterar o arquivo `tarefa.php` para que ele possa receber o arquivo. Faremos a validação e gravação do anexo naquele **if** que verifica se existem dados no post. A estrutura que devemos construir irá verificar se o arquivo foi enviado, depois irá verificar se o arquivo é do tipo pdf ou zip, afinal, estes foram os arquivos que decidimos aceitar. É só então que fará a gravação do nome do arquivo no banco de dados. Este trecho de código deverá ficar assim:

```
<?php
    ...

    if (tem_post()) {
        // upload dos anexos
        $tarefa_id = $_POST['tarefa_id'];

        if (! isset($_FILES['anexo'])) {
            $tem_erros = true;
            $erros_validacao['anexo'] =
                'Você deve selecionar um arquivo para anexar';
        } else {
            if (tratar_anexo($_FILES['anexo'])) {
                $anexo = array();
                $anexo['tarefa_id'] = $tarefa_id;
                $anexo['nome'] = $_FILES['anexo']['name'];
                $anexo['arquivo'] = $_FILES['anexo']['name'];
            } else {
                $tem_erros = true;
                $erros_validacao['anexo'] =
                    'Envie apenas anexos nos formatos zip ou pdf';
            }
        }
```

```
        if (! $tem_erros) {
            gravar_anexo($conexao, $anexo);
        }
    }

    ...
```

Perceba que esta estrutura é bem parecida com a gravação de tarefas, as diferenças estão no uso das duas novas funções: `tratar_anexo()` e `gravar_anexo()`. A função `tratar_anexo()` será responsável por verificar se o anexo está no formato correto e também por copiá-lo para a pasta onde serão armazenados. Já a função `gravar_anexo()` será responsável por cadastrar o anexo no banco de dados, no mesmo estilo da função `gravar_tarefa()`.

Na função `tratar_anexo()` teremos que verificar se o arquivo é do tipo correto, ou seja, pdf ou zip e, quando o tipo for correto, vamos guardar o arquivo em uma pasta.

Guardar arquivos em pastas? E o banco de dados?

Pode parecer estranho, mas não vamos usar o banco de dados para guardar os anexos dos usuários.

O que vamos guardar no banco de dados é apenas o nome do arquivo, assim teremos uma referência para acessar cada arquivo quando necessário.

Já os arquivos serão armazenados em um diretório, uma pasta, no servidor, assim como guardamos arquivos em nossos computadores.

Esta é uma maneira simples e eficiente para lidar com o upload e armazenamento de arquivos. Ela mantém o banco de dados menor, apenas com os nomes dos arquivos e deixa os arquivos para o sistema de arquivos do sistema operacional. Faz sentido, né?

Veja como deverá ficar a função `tratar_anexo()`, que será adicionada no arquivo `ajudantes.php`:

```
<?php
    ...
```

```
function tratar_anexo($anexo) {
    $padrao = '/^.+(\.pdf|\.zip)$/';
    $resultado = preg_match($padrao, $anexo['name']);

    if (! $resultado) {
        return false;
    }

    move_uploaded_file($anexo['tmp_name'],
                       "anexos/{$anexo['name']}");

    return true;
}

...
```

A forma que estamos usando para verificar se o arquivo é do tipo correto é através da extensão. E aqui temos mais um uso importante para as expressões regulares. Desta vez, estamos usando o ponto, que é uma espécie de curinga das expressões regulares e serve para casar com qualquer coisa. Logo após o ponto, temos o sinal de mais, que indica que o ponto, ou seja, qualquer coisa, deve casar pelo menos uma vez e pode casar diversas vezes. Este trecho `.+` já serve para casar qualquer texto, mas no nosso caso queremos apenas os arquivos que terminem com `.pdf` ou `.zip`, por isso, precisamos adicionar os trechos `\.pdf` e `\.zip`. Perceba que nestes trechos queremos que a expressão case o ponto literal. Para tanto, usamos a barra invertida antes dele, para fazer o **escape** do ponto, pois isso significa algo como "ei,expressão regular, aqui eu quero um ponto mesmo, não qualquer coisa, como é o padrão do ponto".

Ainda tem mais um detalhe importante nesta expressão regular, que é o uso do **ou**, pois precisamos que o arquivo seja `zip` **ou** `pdf`. Isso é feito usando os parênteses e o pipe (barra vertical), assim: **(isso|aquilo)**, esta expressão casaria as palavras **isso** ou **aquilo**.

Não falei que as expressões regulares merecem um livro só para elas?

Muito bem, após casar a expressão regular, movemos o arquivo temporário para o seu destino final.

Arquivo temporário? O que tem no $_FILES?

Quando enviamos um arquivo através de um formulário para o PHP, ele cria a super global `$_FILES`, sim no mesmo estilo das super globais `$_GET` e `$_POST`. Cada campo do tipo `file` é colocado em um array dentro de $_FILES, por isso conseguimos chamar a função `tratar_anexo()` passando o índice `anexo`, que é o nome do campo do anexo no nosso formulário.

Cada índice do `$_FILES` é um array que contém os seguintes índices:

- **name**: nome do arquivo;
- **type**: o tipo mime do arquivo;
- **tmp_name**: o nome onde o arquivo foi salvo temporariamente;
- **error**: um número que representa um erro no upload. Zero significa que está tudo certo;
- **8size**: O tamanho do arquivo.

O arquivo enviado para o servidor é gravado em um nome temporário, por isso precisamos pegá-lo e copiar para seu destino final. Veja que a função `tratar_anexo()` está movendo o arquivo temporário para uma pasta chamada **anexos**, que ainda não existe.

Para mover o arquivo para sua localização final usamos a função `move_uploaded_file()`, que faz o que seu nome diz, verificando se o arquivo de origem realmente foi enviado para o PHP através de um formulário na requisição atual.

A pasta de uploads

Estamos enviando os arquivos para uma pasta chamada `anexos` usando a função `move_uploaded_file()` e, para que isso funcione, ela precisa existir. Então, crie uma pasta chamada `anexos` na mesma pasta onde está o arquivo `tarefas.php`, ou seja a `tarefas`.

ATENÇÃO: Usuários de sistemas GNU/Linux e Mac OS X (ou outros baseados em Unix) deverão dar permissão de escrita para o Apache no diretório de upload. Um `chmod 777 anexos/` resolve, mas você também pode mudar o grupo deste diretório e dar permissão para o servidor web.

A função `tratar_anexo()` retorna `true` ou `false`. Caso seja `true`, criamos um array novo chamado `$anexo` e guardamos nele os dados do upload; se for `false` geramos um erro que será exibido no formulário.

10.6 Gravando os dados do anexo no banco dados

Estando tudo certo com o upload, vamos para a gravação dos dados no banco usando a função `gravar_anexo()`, que ficará no arquivo `banco.php`:

```
<?php
    ...

    function gravar_anexo($conexao, $anexo)
    {
        $sqlGravar = "INSERT INTO anexos
            (tarefa_id, nome, arquivo)
            VALUES
            (
                {$anexo['tarefa_id']},
                '{$anexo['nome']}',
                '{$anexo['arquivo']}'
            )
        ";

        mysqli_query($conexao, $sqlGravar);
    }
```

```
    ...
```

Sem muitos segredos, certo? A maior diferença para a gravação dos dados de uma tarefa está no campo `tarefa_id`, que guarda uma referência à tarefa à qual este anexo pertence. Como já falei anteriormente neste capítulo, isso faz parte da normalização de bancos de dados e vale um estudo adicional sobre o assunto.

10.7 Exibindo os anexos

Agora precisamos buscar os anexos no banco para poder fazer uma lista na página com os detalhes da tarefa. Para isso vamos usar uma função nova chamada `buscar_anexos()`, que receberá, como sempre, a conexão e também o **id** da tarefa. Adicione a chamada desta função no final do arquivo `tarefa.php`, logo abaixo da linha que busca os dados da tarefa:

```php
<?php
    ...

    $tarefa = buscar_tarefa($conexao, $_GET['id']);
    $anexos = buscar_anexos($conexao, $_GET['id']);

    include "template_tarefa.php";

    ...
```

Já a função `buscar_anexos()` deverá ser adicionada no arquivo `banco.php`:

```php
<?php
    ...

    function buscar_anexos($conexao, $tarefa_id)
    {
        $sql = "SELECT * FROM anexos WHERE tarefa_id = {$tarefa_id}";
        $resultado = mysqli_query($conexao, $sql);

        $anexos = array();

        while ($anexo = mysqli_fetch_assoc($resultado)) {
            $anexos[] = $anexo;
        }
```

```
        return $anexos;
    }

    ...
```

Quase sem diferenças para a função `buscar_tarefas()`, né? Basicamente, muda só o código SQL que estamos usando. Veja que a consulta SQL filtra apenas as tarefas que contenham o campo `tarefa_id` com o valor da tarefa que queremos exibir. Como o campo `tarefa_id` é uma chave em outra tabela, a tabela `tarefas`, ela é chamada de chave estrangeira. Ops, já falei sobre isso, mas é bom reforçar. :)

Para exibir os anexos, precisamos adicionar uma tabela no arquivo `template_tarefa.php`, logo abaixo do H2 que diz "Anexos" e acima do formulário para cadastrar um anexo. Para criar a tabela vamos usar a variável `$anexos` que contém o resultado da função `buscar_anexos()`:

```
...

<h2>Anexos</h2>
<!-- lista de anexos -->

<?php if (count($anexos) > 0) : ?>
    <table>
        <tr>
            <th>Arquivo</th>
            <th>Opções</th>
        </tr>

        <?php foreach ($anexos as $anexo) : ?>
            <tr>
                <td><?php echo $anexo['nome']; ?></td>
                <td>
                    <a href="anexos/<?php echo $anexo['arquivo']; ?>">
                        Download
                    </a>
                </td>
            </tr>
        <?php endforeach; ?>

    </table>
<?php else : ?>
```

```
    <p>Não há anexos para esta tarefa.</p>
<?php endif; ?>

<!-- formulário para um novo anexo -->

...
```

Veja que estamos verificando se a lista de anexos tem algum índice antes de tentar exibir a lista. Cada item da lista tem um elemento `a` que faz um link para o download do arquivo. Aqui não tem segredos, é um link normal que aponta o arquivo guardado na pasta `anexos`.

Experimente fazer o upload do arquivo, sua página de detalhes da tarefa deverá ficar parecida com esta:

Figura 10.5: Página com os detalhes da tarefa, a lista de anexos e o formulário para cadastrar novos anexos

10.8 Resumo

Neste capítulo tratamos do upload de arquivos usando PHP. Os uploads são colocados dentro da super global `_FILES`. Além disso criamos uma nova tabela no banco e usamos mais um pouco de expressões regulares para validar o tipo de arquivo enviado.

10.9 Desafios

Depois de trabalhar duro no envio de arquivos para as tarefas, vamos para mais alguns desafios!

- Na lista de contatos (sim, aquela dos desafios anteriores): 1) adicione uma página para exibir os detalhes de cada contato; 2) adicione o upload de fotos para a lista de contatos. Lembre-se de validar se o arquivo é uma imagem, como `jpg`, `png` etc.; 3) adicione a foto do contato na página de detalhes.
- No projeto do estacionamento, adicione uma opção para upload da foto do carro no momento em que entrou e no momento em que saiu do estacionamento. Isso é uma garantia que os clientes pediram, pois alguns disseram que seus veículos foram amassados dentro do estacionamento.

Capítulo 11

Lembretes de tarefas por e-mail

É bem comum que diversos tipos de aplicações web enviem e-mails para os usuários com algum tipo de informação e também lembretes. Por isso vamos adicionar uma opção em nossa lista de tarefas para enviar e-mails com lembretes das tarefas.

Este não será o tipo de lembrete que é enviado automaticamente no dia definido como prazo para a tarefa. O que faremos é a adição de um checkbox **Enviar lembrete** nos formulários de criação e edição de tarefas e, quando esta opção estiver marcada, vamos enviar o e-mail para um endereço de e-mail que deixaremos cadastrado no código.

O PHP possui algumas opções padrão para envio de e-mails, mas desta vez vamos usar uma biblioteca adicional que facilitará nosso trabalho.

11.1 Definindo o e-mail de aviso

O e-mail de aviso será um HTML simples com os dados da tarefa. A ideia é mandar por e-mail uma página bem parecida com a página de detalhes e, quando necessário,

adicionar também os anexos da tarefa ao e-mail. E é aqui que a biblioteca de envio de e-mails será bastante útil!

O corpo do nosso e-mail ficará em um novo arquivo que vamos chamar de `template_email.php`. Este arquivo deverá ficar junto com os demais dentro do diretório `tarefas`, e seu conteúdo será apenas este:

```
<h1>Tarefa: <?php echo $tarefa['nome']; ?></h1>

<p>
    <strong>Concluída:</strong>
    <?php echo traduz_concluida($tarefa['concluida']); ?>
</p>
<p>
    <strong>Descrição:</strong>
    <?php echo nl2br($tarefa['descricao']); ?>
</p>
<p>
    <strong>Prazo:</strong>
    <?php echo traduz_data_para_exibir($tarefa['prazo']); ?>
</p>
<p>
    <strong>Prioridade:</strong>
    <?php echo traduz_prioridade($tarefa['prioridade']); ?>
</p>

<?php if (count($anexos) > 0) : ?>
    <p><strong>Atenção!</strong> Esta tarefa contém anexos!</p>
<?php endif; ?>
```

Repare que este template para o e-mail é bem parecido com a página de detalhes da tarefa, aquela que tem o formulário para adicionar anexos. A diferença é que aqui não precisamos exibir o link para voltar para a lista de tarefas e também não precisamos exibir os anexos, apenas uma informação de que existem anexos para a tarefa.

Agora vamos deixar esse arquivo guardado um pouco e vamos ajustar outros pontos da aplicação. Certo, eu sei que é chato escrever código e não ver o funcionamento, mas achei melhor já deixarmos este arquivo pronto agora, para não atrapalhar o raciocínio mais para frente.

11.2 Unificando a configuração da aplicação com constantes

Até agora nossa aplicação não tem muitos itens configuráveis, na verdade temos apenas os dados de conexão com o banco de dados. Mas, a partir de agora, teremos uma configuração para definir o endereço de e-mail para onde os e-mails de aviso serão enviados.

Tudo bem, ainda são poucas informações, mas, acredite, é melhor começar a organização ainda pequeno, pois facilita o crescimento.

Muito bem, vamos criar então um novo arquivo para manter as configurações da nossa aplicação. Ele será o `config.php` e deverá ser colocado com os demais arquivos na pasta `tarefas`. Inicialmente, ele deverá conter os dados de conexão ao banco e também o e-mail de aviso das tarefas. Veja como deverá ficar seu conteúdo:

```
<?php

// Conexão ao banco de dados (MySQL)
define("BD_SERVIDOR", "127.0.0.1");
define("BD_USUARIO", "sistematarefa");
define("BD_SENHA", "sistema");
define("BD_BANCO", "tarefas");

// E-mail para notificação
define("EMAIL_NOTIFICACAO", "meuemail@meudominio.com.br");
```

Neste arquivo estamos usando a função `define()` que serve para definir **constantes** no PHP. A função `define()` recebe o nome da constante e o seu valor. Uma constante é um identificador para um valor que não será alterado durante a execução da aplicação.

Fica fácil entender as constantes quando já conhecemos bem as variáveis, que são identificadores para valores que podem mudar, por isso têm o nome de **variáveis**. Para exibir as constantes usamos apenas seus nomes, sem o cifrão na frente, como as variáveis. Veja um exemplo:

```
<?php
    $linguagem = "PHP";
    define("BANCO", "MySQL");

    echo "Eu uso {$linguagem} com o banco " . BANCO;
    // Exibe: Eu uso PHP com o banco MySQL
```

Repare que estou deixando as constantes sempre em caixa alta, ou seja, todas as letras em maiúsculo. Isso não é obrigatório, mas é um padrão que o pessoal do PHP segue, pois desta maneira fica fácil de identificar onde elas estão em um código.

No nosso arquivo `banco.php` definimos quatro variáveis para guardar os dados de acesso ao banco de dados. Mas esses valores não deverão mudar durante o uso da aplicação, por isso eles poderiam ficar em constantes. Agora, por exemplo, a variável `$tarefa` recebe novos dados e que podem ser alterados, por isso não pode ser uma constante. Aliás, as constantes não podem ser definidas como **arrays**! Apenas valores como números e strings.

Vamos então alterar o arquivo `banco.php` para usar nossas novas constantes para a comunicação com o banco. Apague a definição das variáveis com os dados de acesso ao banco e mude a chamada da função `mysqli_connect()` para usar as constantes:

```
<?php

$conexao = mysqli_connect(BD_SERVIDOR, BD_USUARIO, BD_SENHA, BD_BANCO);

...
```

Veja como uma rápida olhada no código já nos revela que estamos usando constantes, graças ao uso da caixa alta.

Se você tentar acessar sua aplicação agora, vai encontrar alguns erros dizendo que os identificadores usados não foram definidos, pois ainda não adicionamos o novo arquivo `config.php` na aplicação.

Nossa aplicação tem diversos pontos de entrada, que são os arquivos que estamos chamando diretamente pelo endereço no navegador. O arquivo `tarefas.php`, por exemplo, é um ponto de entrada, pois ele é chamado pelo navegador. Já o `template.php` é incluído por outros arquivos, então não é um ponto de entrada. Precisamos alterar todos os pontos de entrada para incluir o arquivo de configuração. Então, adicione a linha a seguir no início dos arquivos `tarefas.php`, `tarefa.php`, `editar.php` e `remover.php`:

```
<?php

    include "config.php"; // Esta é a nova linha
    include "banco.php";

    ...
```

Lembre-se que o arquivo `banco.php` depende das constantes definidas no arquivo `config.php`, então este deve ser incluído primeiro.

11.3 Adicionando a opção de aviso por e-mail

O usuário precisa informar que quer receber por e-mail um lembrete da tarefa, por isso precisamos alterar o formulário, adicionando um checkbox com a opção de envio do e-mail.

Vamos alterar o arquivo `formulario.php` para adicionar a nova opção logo abaixo da opção de tarefa concluída:

```
...

<label>
    Tarefa concluída:
    <input type="checkbox" name="concluida" value="1"
        <?php echo ($tarefa['concluida'] == 1) ? 'checked' : ''; ?>
    />
</label>
<label>
    Lembrete por e-mail:
    <input type="checkbox" name="lembrete" value="1" />
</label>

...
```

Veja que é apenas um checkbox HTML com o nome `lembrete` e o valor `1`. Este campo será verificado durante a gravação e durante a edição de uma tarefa e, caso seja necessário, o e-mail com o aviso será disparado.

Vamos fazer a alteração primeiro na gravação de novas tarefas, que fica no arquivo `tarefas.php`. O envio do e-mail será feito usando a função `enviar_email()`, que ainda não existe. Essa função receberá como parâmetros apenas os dados da tarefa e os dados dos anexos, sendo que nem sempre uma tarefa terá anexos, então o parâmetro dos anexos será opcional. Opcional? Calma, veremos isso logo mais.

A nossa primeira alteração será na gravação da tarefa, então não precisaremos enviar os dados dos anexos, pois nossa aplicação só permite adicionar anexos na página de detalhes da tarefa.

No arquivo `tarefas.php`, logo após a gravação da tarefa usando a função `gravar_tarefa()`, adicione um **if** que verifica que a opção de lembrete foi marcada e então chama a função `enviar_email()`:

```
<?php
    ...

    if (! $tem_erros) {
        gravar_tarefa($conexao, $tarefa);

        if (isset($_POST['lembrete']) && $_POST['lembrete'] == '1') {
            enviar_email($tarefa);
        }

        header('Location: tarefas.php');
        die();
    }

    ...
```

Veja que a função `enviar_email()` está sendo chamada com apenas um parâmetro, que são os dados da tarefa.

Agora, altere também o arquivo `editar.php`. Essa alteração será bem parecida com a anterior, mas aqui precisamos mandar também os dados dos anexos. Então, fica assim:

```
<?php
    ...

    if (! $tem_erros) {
        editar_tarefa($conexao, $tarefa);

        if (isset($_POST['lembrete']) && $_POST['lembrete'] == '1') {
            $anexos = buscar_anexos($conexao, $tarefa['id']);

            enviar_email($tarefa, $anexos);
        }

        header('Location: tarefas.php');
        die();
    }
```

```
...
```

Como já temos a lógica para pegar os anexos dentro de uma função, basta chamá-la e pegar seu resultado para termos os anexos.

Agora que já preparamos a aplicação para enviar os e-mails, vamos finalmente para a função `enviar_email()`!

11.4 A função enviar_email()

Antes de escrever a função para enviar e-mails, vamos pensar no que é necessário fazer para enviar um e-mail usando um sistema como o GMail outros webmails. Vou colocar aqui o passo a passo do que é necessário para enviar um e-mail:

- Acessar o sistema de e-mails;
- Fazer a autenticação com usuário e senha;
- Usar a opção para escrever um e-mail;
- Digitar o e-mail do destinatário;
- Digitar o assunto do e-mail;
- Escrever o corpo do e-mail;
- Adicionar os anexos quando necessário;
- Usar a opção de enviar o e-mail.

Nossa! Uma tarefa tão simples como enviar um e-mail virou uma atividade dividida em oito passos! Então, para enviar um e-mail usando PHP deve ser algo bem parecido. Sendo assim vamos criar a função para enviar os e-mails usando um monte de comentários, que vão nos guiar para a implementação do código. No arquivo `ajudantes.php`, adicione a função `enviar_email()`:

```
<?php
    ...

    function enviar_email($tarefa, $anexos)
    {
```

```
        // Acessar o sistema de e-mails;
        // Fazer a autenticação com usuário e senha;
        // Usar a opção para escrever um e-mail;
        // Digitar o e-mail do destinatário;
        // Digitar o assunto do e-mail;
        // Escrever o corpo do e-mail;
        // Adicionar os anexos quando necessário;
        // Usar a opção de enviar o e-mail.
    }
```

Agora já temos uma espécie de mapa que irá nos guiar. Mas, antes de seguir o mapa e fazer a função realmente mandar e-mails, repare nos parâmetros que esta função pode receber. Veja que temos uma variável `$tarefa` e uma variável `$anexos`. O problema que começa a nos perturbar é saber que após a gravação da tarefa estamos chamando essa função com apenas um parâmetro, enquanto que na edição estamos passando os dois. Isso certamente vai gerar um erro, pois precisamos fornecer os parâmetros obrigatórios de uma função.

Parâmetros de funções

Em geral as funções recebem parâmetros e fazem algum tipo de trabalho repetitivo com eles. Mas algumas vezes podemos precisar de funções que podem receber um certo parâmetro ou podem ter algum tipo de valor padrão para um.

Este é exatamente o nosso caso na função `enviar_email()`, pois o que realmente será necessário sempre são os dados das tarefas, afinal não podemos enviar um e-mail sem isso. Já os anexos serão opcionais, já que podemos muito bem ter tarefas sem anexos.

Para fazer com que um parâmetro seja opcional em uma função PHP basta atribuir um valor logo após a declaração do parâmetro na função:

```
function pular_linha($c = '<br />') { echo $c; }
```

A função `pular_linha()` por padrão pula uma linha no HTML usando a tag **
, mas poderíamos trocar o comportamento padrão chamando a função e falando para pular linha em um arquivo txt usando o caractere **\n:

```
pular_linha("\n");
```

Como a lista de anexos é um array, podemos alterar a declaração da função `enviar_email()` para ter uma lista de anexos vazia por padrão:

```
<?php
    ...

    function enviar_email($tarefa, $anexos = array())
    {
        ...
    }

    ...
```

Agora vamos voltar à lista de passos para enviar o e-mail. Praticamente a lista depende de algum tipo de comunicação com o sistema de e-mails, menos o item **escrever o corpo do e-mail**, e até já temos o arquivo `template_email.php` com o template. Então, vamos começar por este item.

11.5 Escrevendo o corpo do e-mail usando um arquivo com o template

Para colocar o texto do corpo do e-mail efetivamente dentro do e-mail, precisamos usar alguma forma de ler o arquivo `template_email.php`, colocando o seu conteúdo já processado (com os valores das variáveis etc.) dentro de uma variável. É alguma coisa parecida com isso:

```
<?php
    ...

    function enviar_email($tarefa, $anexos = array())
    {
        $corpo = include "template_email.php";
    }
```

O arquivo `template_email.php` precisa das variáveis `$tarefa` e `anexos`, que já estão sendo recebidas pela função. A ideia aqui é que a variável `$corpo` fique com o HTML resultante do processamento do arquivo `template_email.php`.

Infelizmente esse código não irá funcionar e o conteúdo do arquivo `template_email.php` será enviado para o navegador.

Para resolver este problema vamos separar essa lógica de ler o arquivo `template_email.php` em uma nova função, pois assim vai ficar mais simples ler e entender e função `enviar_email()`:

```
<?php
    ...

    function enviar_email($tarefa, $anexos = array())
    {
        $corpo = preparar_corpo_email($tarefa, $anexos);

        ...
    }

    function preparar_corpo_email($tarefa, $anexos)
    {
        // Aqui vamos pegar o conteúdo processado do template_email.php
    }
```

Agora que separamos a lógica para preparar o corpo do e-mail, vamos pensar em como isso deve ser feito. O nosso maior problema é que, se incluirmos um arquivo com HTML usando o **include**, seu conteúdo será enviado para o navegador, então temos que fazer um passo a passo dessa forma:

- Falar para o PHP que não é para enviar o processamento para o navegador;
- Incluir o arquivo `template_email.php`;
- Guardar o conteúdo do arquivo em uma variável;
- Falar para o PHP que ele pode voltar a mandar conteúdos para o navegador.

Estamos cheios de listas descrevendo os procedimentos neste capítulo, heim? Mas é por uma boa razão, já que isso ajuda a pensar primeiro na lógica e depois na implementação usando PHP.

Para fazer a função `preparar_corpo_email()` vamos usar algumas funções do PHP que fazem exatamente o que foi descrito na lista anterior. Essas funções impedem o PHP de enviar conteúdos para o navegador e nos permitem pegar esses conteúdos e guardar em variáveis. Veja como fica simples a implementação:

```
<?php
    ...

    function preparar_corpo_email($tarefa, $anexos)
    {
        // Aqui vamos pegar o conteúdo processado do template_email.php

        // Falar para o PHP que não é para enviar o processamento para o
        // navegador:
        ob_start();

        // Incluir o arquivo template_email.php:
        include "template_email.php";

        // Guardar o conteúdo do arquivo em uma variável;
        $corpo = ob_get_contents();

        // Falar para o PHP que ele pode voltar a mandar conteúdos para o
        // navegador.
        ob_end_clean();

        return $corpo;
    }
```

Eu deixei os comentários de propósito, pois eles são a mesma lista de passos que fizemos para montar a lógica. Depois você pode apagá-los.

Na função `preparar_corpo_email()`, usamos a função `ob_start()`, que coloca uma espécie de cancela no fluxo que o PHP envia para o navegador. Depois fizemos a inclusão do arquivo `template_email.php`, aquele que fizemos lá no começo deste capítulo, usando a instrução **include**. Aí vem uma parte bem mágica em que usamos a função `ob_get_contents()`, que faz algo como pegar todo mundo que estava esperando a cancela abrir para passar e retorna esse valor para a variável `$corpo`. Por último, usamos a função `ob_end_clean()` para finalizar o processo, que é tipo reabrir a cancela e deixar os novos dados passarem para o navegador.

Complicado? Sim, um pouquinho, mas é só treinar para dominar essa técnica. Se não fosse por esta técnica teríamos que fazer algo como uma variável de texto com todo o HTML necessário e as variáveis.

11.6 Instalando uma biblioteca para enviar e-mails

Nossa função `enviar_email()` já tem o corpo do e-mail, mas ainda tem vários passos que ela não faz e que serão feitos com a ajuda de uma biblioteca para e-mails.

Existem várias bibliotecas para envio de e-mails usando PHP, inclusive algumas opções embutidas no PHP. A ideia de usar uma biblioteca adicional neste livro é mostrar como isso pode ser feito, além de suprir alguns pontos que seriam bem trabalhosos de fazer com as opções padrão, como anexar arquivos.

A biblioteca escolhida é a PHPMailer. Para baixá-la acesse https://github.com/PHPMailer/PHPMailer e clique no botão `Download ZIP`, conforme a imagem a seguir:

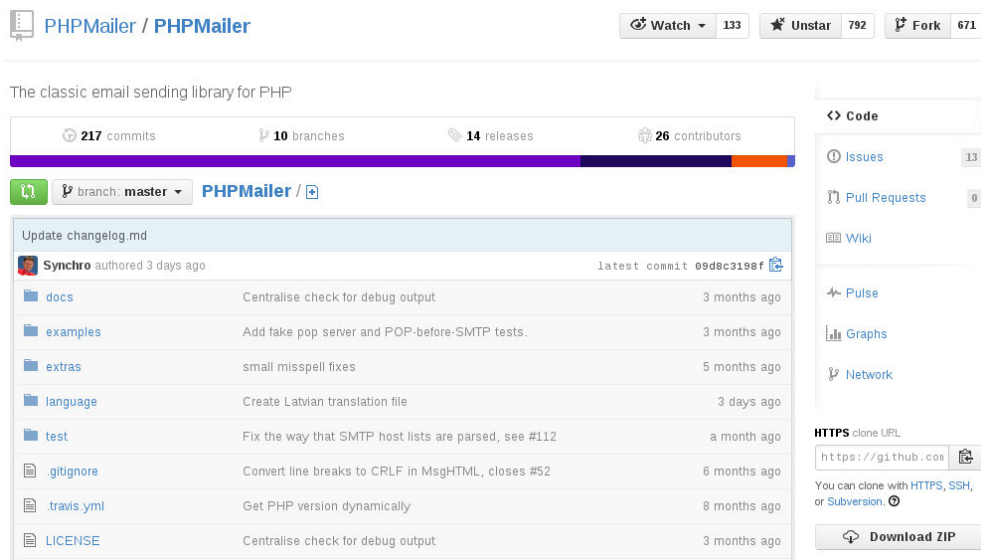



Figura 11.1: Baixando o PHPMailer no GitHub. A opção está no canto inferior direito

Após baixar, crie um novo diretório, uma pasta chamada `bibliotecas`, dentro da pasta `tarefas`, e descompacte o conteúdo do zip dentro dela. Após descompactar você verá uma nova pasta chamada `PHPMailer-master`. Renomeie-a para apenas `PHPMailer`. Depois de renomear você vai ficar com uma estrutura de pastas assim:

`tarefas > bibliotecas > PHPMailer`

Vale lembrar que nosso projeto está dentro da pasta `tarefas`.

Agora podemos voltar para a função `enviar_email()`.

11.7 Finalizando a função enviar_email()

Para finalizar a função `enviar_email()`, precisamos usar a biblioteca PHPMailer. Veremos agora algumas linhas de código um pouco diferentes do que já vimos até aqui, pois a biblioteca PHPMailer usa um negócio chamado **Programação Orientada à Objetos**.

De forma simplificada podemos pensar que um **objeto** é uma estrutura que pode ter várias variáveis dentro dela, assim como os arrays, e também pode ter funções dentro dela, que servem para manipular suas variáveis.

Lembra da lista de passos necessários para se enviar um e-mail? Vamos agora preencher esta lista usando as opções da PHPMailer. Vou colocar o código passo a passo, depois juntamos tudo na função.

O primeiro passo é incluir a PHPMailer:

```
<?php
    ...

    include "bibliotecas/PHPMailer/PHPMailerAutoload.php";
```

Os próximos passos são o acesso ao sistema de e-mails, a autenticação e a opção de escrever um novo e-mail:

```
<?php
    ...

    // Acessar o sistema de e-mails;
    // Fazer a autenticação com usuário e senha;
    // Usar a opção para escrever um e-mail;

    $email = new PHPMailer(); // Esta é a criação do objeto

    $email->isSMTP();
    $email->Host = "smtp.gmail.com";
    $email->Port = 587;
    $email->SMTPSecure = 'tls';
    $email->SMTPAuth = true;
    $email->Username = "meuemail@gmail.com";
```

```
    $email->Password = "minhasenha";
    $email->setFrom("meuemail@gmail.com", "Avisador de Tarefas");
```

Neste último trecho, bastante coisa aconteceu! Primeiro, criamos o **objeto** `$email` usando a classe `PHPMailer`. Não se preocupe com isso ainda, apenas entenda que no objeto `$email` tem variáveis e funções embutidas. Acessamos estas variáveis e funções usando a seta, que é o sinal de menos seguido por um sinal de maior: `->`. Identificamos o que é variável e o que é função usando os parênteses nas funções.

Como eu quero enviar o e-mail usando o GMail, precisei dizer que o tipo de conexão é `SMTP`, o endereço do servidor é `smtp.gmail.com`, a porta para conexão é a `587` e que ele tem que usar a criptografia `tls`. Além disso, também falei que é necessário autenticar no SMTP usando o usuário (Username) e senha (Password) informados. Por último, eu coloquei o `From` do e-mail, para dizer de onde o e-mail partiu.

O próximo passo é o destinatário usando a nossa constante `EMAIL_NOTIFICACAO`:

```
<?php
    ...

    // Digitar o e-mail do destinatário;
    $email->addAddress(EMAIL_NOTIFICACAO);
```

Agora tem o assunto do e-mail:

```
<?php
    ...

    // Digitar o assunto do e-mail;
    $email->Subject = "Aviso de tarefa: {$tarefa['nome']}";
```

Tem também o corpo do e-mail, que vamos enviar usando HTML para deixar formatado:

```
<?php
    ...

    // Escrever o corpo do e-mail;
    $corpo = preparar_corpo_email($tarefa, $anexos);
    $email->msgHTML($corpo);
```

Os anexos, quando existirem, são os próximos:

```
<?php
    ...

    // Adicionar os anexos quando necessário;
    foreach ($anexos as $anexo) {
        $email->addAttachment("anexos/{$anexo['arquivo']}");
    }
```

Veja que o `foreach` foi usado para os anexos. Caso o array `$anexos` esteja vazio, ele simplesmente não entra no laço.

Agora só falta a opção de enviar o e-mail! Veja como fica:

```
<?php
    ...

    // Usar a opção de enviar o e-mail.
    $email->send();
```

Ufa! Esse é todo o código necessário para enviar o e-mail! Se pensar bem, é tipo o passo a passo para enviar um e-mail, apenas codificamos em PHP. Veja como fica a função `enviar_email()` completa e sem os comentários:

```
<?php

    function enviar_email($tarefa, $anexos = array())
    {
        include "bibliotecas/PHPMailer/PHPMailerAutoload.php";

        $corpo = preparar_corpo_email($tarefa, $anexos);

        $email = new PHPMailer();

        $email->isSMTP();
        $email->Host = "smtp.gmail.com";
        $email->Port = 587;
        $email->SMTPSecure = 'tls';
        $email->SMTPAuth = true;
        $email->Username = "meuemail@email.com";
        $email->Password = "minhasenha";
        $email->setFrom("meuemail@email.com", "Avisador de Tarefas");
```

```
    $email->addAddress(EMAIL_NOTIFICACAO);
    $email->Subject = "Aviso de tarefa: {$tarefa['nome']}";
    $email->msgHTML($corpo);

    foreach ($anexos as $anexo) {
        $email->addAttachment("anexos/{$anexo['arquivo']}");
    }

    $email->send();
}
```

A diferença é que eu coloquei o uso da função `preparar_corpo_email()` logo no começo.

Agora já podemos usar a opção de avisar por e-mail! Eu editei uma tarefa que tem anexos e marquei a opção para avisar:

Figura 11.2: Editando uma tarefa e marcando a opção de lembrete

E veja só como o e-mail chegou para mim:

Figura 11.3: E-mail de aviso no GMail, com anexo

O uso da PHPMailer facilita bastante o trabalho para envio de e-mails, pois reduz praticamente ao passo a passo seguido para enviar um usando um webmail, sem maiores complicações.

Não uso o GMail

O pessoal do PHPMailer tem alguns exemplos de como fazer a parte da conexão usando outros servidores além do GMail. Para saber mais acesse https://github.com/PHPMailer/PHPMailer/tree/master/examples

Dica importante!

Evite usar o seu e-mail pessoal para enviar e-mails em uma aplicação como a nossa, pois seu usuário e senha ficam expostos no código.

Faça uma conta de e-mail para isso, ou se você estiver desenvolvendo algo na empresa onde trabalha, veja com o seu administrador de redes e servidores como deve ser a autenticação para mandar e-mails (ou se realmente precisa autenticar).

11.8 Resumo

Este capítulo começou com uma ideia simples: mandar e-mails com lembretes das tarefas. Vimos como instalar e usar uma biblioteca com orientação a objetos para envio de e-mails, trabalhamos com constantes e com a função `define()` para centralizar a configuração da aplicação. Além disso, usamos os controles de saída do PHP com as funções `ob_start()`, `ob_get_contents()` e `ob_end_clean()` para conseguir fazer a inclusão de um template com o resultado em uma variável.

11.9 Desafios

É claro que este capítulo também tem alguns desafios.

- Ter quatro pontos de entrada dificultou um pouco a adição do arquivo de configuração. Faça uma cópia do projeto até aqui e pense em uma forma de manter apenas um ponto de entrada que consiga tratar todos os casos.
- O arquivo de configuração tem apenas o e-mail de destino. Adicione constantes para configurar o do remetente, o usuário e a senha para autenticar, e o nome que aparece no "from".
- Na lista de contatos, adicione a opção de mandar os dados de um contato para um endereço de e-mail.
- Ainda na lista de contato, adicione a foto do contato como anexo do e-mail.
- Mais um para a lista de contatos: centralize as configurações usando constantes em um arquivo separado.
- No sistema do estacionamento, adicione a opção de mandar os dados de uma estadia para um e-mail, mandando também as fotos do veículo.

Em todos os desafios, procure usar a técnica de ler o template do e-mail usando as funções `ob_start()` etc.

Capítulo 12

Hospedagem de aplicações PHP

Desenvolver aplicações web pode ser bastante divertido e útil, pois podemos desenvolver ferramentas que solucionam problemas que temos em nosso dia a dia, como foi o caso do nosso sistema de tarefas desenvolvido durante os capítulos deste livro. O problema é que até este momento ficamos presos a nossos computadores, pois nossas aplicações funcionam apenas neles. Quando acessamos um site, um blog ou outra aplicação web, estamos acessando algo que foi desenvolvido assim como a nossa aplicação, mas que está localizado em um servidor na internet. Quando colocamos nossa aplicação em um servidor na internet, chamamos isso de **hospedagem**.

Quando nossa aplicação está em hospedada em um servidor, ela pode ser acessada a partir de qualquer computador também conectado à internet, ou até mesmo de tablets e smartphones!

12.1 Sua aplicação para o mundo!

Bem, na verdade ainda não é para o mundo, é apenas para você, mas você poderá acessar estando em qualquer lugar, bastando estar conectado à internet. Digo que é

apenas para você pois nossa lista de tarefas não tem separação por usuários, então serve para apenas um usuário.

Para colocarmos nossa aplicação online precisamos encontrar um servidor para hospedagem que suporte nossas necessidades. Depois teremos que configurar como será o funcionamento da aplicação no servidor e então poderemos enviar nossos arquivos para lá.

12.2 Escolhendo um servidor para hospedagem

Para publicar uma aplicação na web precisamos utilizar um servidor que tenha suporte às tecnologias que estamos usando. No nosso caso é o PHP e o MySQL. Existem diversas opções para hospedar aplicações PHP e MySQL, desde empresas que oferecem o serviço gratuitamente (mas podem adicionar algum tipo de propaganda), até opções para quem quer gerenciar o servidor, instalar as ferramentas necessárias e manter tudo funcionando.

No nosso caso, precisamos dos tipos mais básicos, pois estamos usando apenas para aprendizado e não teremos um grande volume de acesso. Mas também é interessante conhecermos uma opção mais avançada, por isso vou demonstrar dois serviços de hospedagem, um mais simples e grátis e outro com mais recursos que oferece um período grátis para experimentar.

Os serviços de hospedagem demonstrados serão a hospedagem da Hostinger e o Jelastic da Locaweb:

- Hostinger http://www.hostinger.com.br/
- Jelastic da Locaweb http://www.locaweb.com.br/produtos/jelastic-cloud.html

12.3 Hospedagem com a Hostinger

A hospedagem com a Hostinger segue um padrão de acesso via **FTP** e uso de um painel de configuração bastante usado em diversos servidores que é o **CPanel**.

O acesso via FTP é bastante utilizado pois ele cria uma conexão que exibe os arquivos remotos e seus arquivos locais de uma maneira que fica simples para copiar de um para o outro. FTP significa *File Transfer Protocol*, ou Protocolo para Transmissão de Arquivos.

O CPanel é uma ferramenta para administração de servidores via navegador. Esta ferramenta possui diversas opções de configuração e é bastante simples de utilizar, por isso é uma opção com um número bem grande de usuários.

12.4 Criação da conta na Hostinger

Para criar a conta para hospedagem de uma aplicação na Hostinger acesse o endereço http://www.hostinger.com.br/ e você verá uma tela parecida com esta:

Figura 12.1: Site da Hostinger

Um formulário de cadastro será apresentado. Preencha os dados e clique no botão `Criar conta`:

Figura 12.2: Cadastro na Hostinger

A próxima tela, após o cadastro, pede para que você verifique o seu e-mail e clique no link de ativação:

Figura 12.3: Pedido de confirmação do cadastro por e-mail

Após clicar no link que chegou por e-mail, você será direcionado para uma página para selecionar um plano de hospedagem. A opção `Gratuito` vai servir bem para testes e até para pequenos sites:

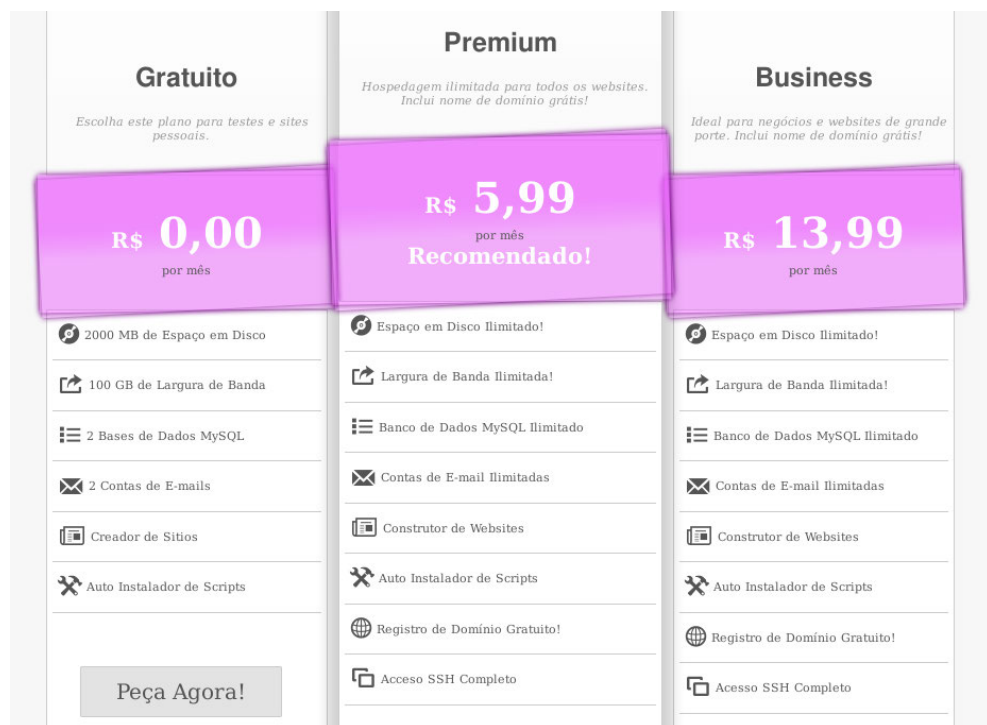



Figura 12.4: Escolha a opção Gratuito para os testes

Na próxima página, você deve criar uma nova conta. Eu escolhi um subdomínio gratuito e aconselho esta opção para os testes:

Criar uma Nova Conta

Subdomínio Gratuito Domínio Próprio

Subdomínio www. tarefasphp .zz.mu

Senha ●●●●●●●●●●●●● Gerar senha

Senha novamente ●●●●●●●●●●●●●

Insira código captcha *

Criar

Figura 12.5: Escolha um subdomínio ou um domínio

Ele pode demorar um pouco para ativar a conta e espalhar o novo endereço pela internet (propagar o DNS). Então você verá esta página quando terminar o cadastro da conta:

Figura 12.6: Aguardando a propagação do DNS

Eu aguardei uns 5 minutos e atualizei a página, e a conta já estava ativa:

Figura 12.7: Como é um subdomínio, ativou bem rápido

Está vendo o botão `Painel de Controle` na tela anterior? Vamos precisar dele logo mais.

E finalmente teremos a conta criada e até o endereço já funcionando! Aqui eu criei um subdomínio no endereço `tarefasphp.zz.mu`. Ao acessá-lo temos a página padrão da Hostinger:

Figura 12.8: Página padrão da Hostinger

Voltando na página das contas, clique no botão `Painel de Controle`. Ele exibirá diversos blocos com áreas diferentes da administração. Navegue até o bloco intitulado `Avançado` e clique na opção `Bases de Dados MySQL`:

Figura 12.9: Use a opção Bases de Dados MySQL

Nesta página será exibido um bloco chamado `Criar Novo Banco de Dados e Usuário MySQL`. Ele já coloca um prefixo para o nome do banco de dados e do usuário, então coloque o nome do banco e o usuário, depois a senha para acesso e confirme:

Criar Novo Banco De Dados e Usuário MySQL

Nome do banco de dados MySQL: u576572591_ taref
Nome de Usuário MySQL: u576572591_ taref
Senha: Gerar senha
Senha novamente:
Criar

Figura 12.10: Criando um novo banco e um usuário MySQL

A página será atualizada e um bloco com os dados de acesso será exibido:

Lista de Banco de Dados e Usuários MySQL Atuais

Banco de Dados MySQL	Usuário MySQL	MySQL Host	Uso de Disco, MB	Ações
u576572591_taref	u576572591_taref	mysql.hostinger.com.br	0.02	Apagar Reparar Bacl phpMyAdmin

Figura 12.11: Os dados de acesso ao MySQL

Guarde esses dados, pois a aplicação vai precisar deles.

Agora devemos voltar ao painel `Avançado` e usar a opção `PHPMyAdmin`, que é o mesmo PHPMyAdmin que usamos em nosso ambiente local. Ele vai pedir o usuário e senha, então coloque o usuário que você acabou de criar e terá acesso ao PHPMyAdmin. Agora siga os passos para criação das tabelas dos capítulos 6 e 10.

Já temos o servidor pronto para receber a aplicação, então precisamos deixar a aplicação pronta para o servidor.

12.5 Configurando a aplicação para a Hostinger

Agora precisamos alterar a aplicação para que funcione no servidor. Na verdade não é tanto trabalho, é apenas a configuração de acesso ao banco de dados. Altere o arquivo `config.php` e coloque os dados de acesso:

```
<?php

    define("BD_SERVIDOR", "mysql.hostinger.com.br");
    define("BD_USUARIO", "seu_usuario");
    define("BD_SENHA", "sua_senha");
    define("BD_BANCO", "seu_banco_taref");
```

Não esqueça de conferir os dados de acesso!

Agora podemos, finalmente, enviar a aplicação para o servidor usando o FTP.

12.6 Enviando a aplicação para a Hostinger

Para enviar os arquivos usando FTP você vai precisar de um software cliente FTP. Existem diversas opções, mas eu recomendo o FileZilla, que pode ser obtido em https://filezilla-project.org/:

Figura 12.12: Site do FileZilla

Use a opção `Download FileZilla Client` e na página de download escolha a opção para o seu sistema operacional. Siga os passos da instalação usando as opções padrão.

> **FileZilla no Linux**
>
> Procure no gerenciador de pacotes da sua distribuição pelo FileZilla. Usuários de Debian/Ubuntu podem instalar usando o apt:
>
> `sudo apt-get install filezilla`

Antes de abrir o FileZilla precisamos dos dados de acesso FTP. Para consegui-los, vá ao CPanel da Hostinger e procure pelo bloco `Arquivos` e, nele, clique na opção `Acesso FTP`:

Figura 12.13: Use a opção Acesso FTP

Você verá a página de acesso FTP:

Acesso FTP	
Host FTP	tarefasphp.zz.mu
IP do FTP	185.28.21.77
Porta FTP	21
Nome de usuário FTP	u576572591
Senha FTP	••••••••••
Pasta para onde os arquivos serão enviados	public_html
Esqueceu sua senha de FTP?	Mudar senha de conta
Clientes FTP recomendados.	SmartFTP or FileZilla

Figura 12.14: Dados de acesso via FTP

É necessário usar a opção `Mudar senha de conta` para escolher uma senha de acesso.

Agora abra o FileZilla e coloque os dados de acesso FTP exibidos pelo CPanel da Hostinger:

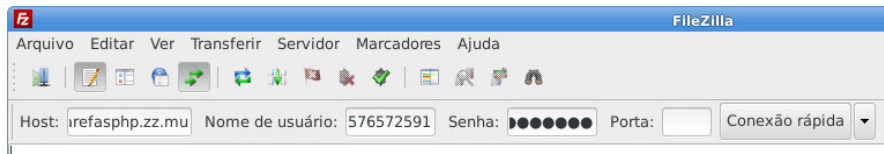



Figura 12.15: Preenchendo os dados de acesso FTP no FileZilla

Após preencher os dados clique em `Conexão rápida`. O FileZilla tem dois paineis, um à esquerda com os seus arquivos locais e um à direita com os arquivos no servidor. No caso da Hostinger será exibido um único arquivo `default.php`. Agora podemos navegar até a pasta onde estão nossos arquivos locais, e então basta clicar e arrastá-los para o painel com os arquivos do servidor. O FileZilla irá iniciar a transmissão, e quando terminar os arquivos serão exibidos no servidor também.

Após finalizar a transmissão dos arquivos podemos acessar o sistema de tarefas já hospedado. O meu ficou hospedado em http://tarefasphp.zz.mu/tarefas.php.

12.7 Hospedagem no Jelastic da Locaweb

O Jelastic é um serviço bem interessante da Locaweb pois nos permite configurar aplicações em caminhos diferentes, ter bancos de dados separados e pacotes com as diferentes versões da aplicação, hospedagem de outras tecnologias além do PHP e MySQL, configuração do PHP e do MySQL etc.

O Jelastic é um pouco mais complexo do que a Hostinger, mas é uma opção interessante para quem deseja ter mais controle sobre as opções do servidor.

12.8 Criação da conta no Jelastic da Locaweb

Para criar uma conta e hospedar uma aplicação no Jelastic acesse o endereço http://www.locaweb.com.br/produtos/jelastic-cloud.html e você verá uma página como esta:

Figura 12.16: Site do Jelastic da Locaweb

Preencha o seu e-mail e clique em `quero experimentar`. Após o processamento, você verá a mensagem de sucesso na criação da conta:

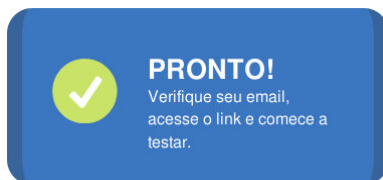



Figura 12.17: Sucesso na criação da conta

Verifique sua caixa de entrada e você deverá ter recebido um e-mail com o endereço e os dados de acesso para configuração do ambiente de hospedagem:

Bem vindo ao Jelastic Cloud!

Olá,

Obrigado por seu interesse em conhecer o Jelastic Cloud Locaweb.

Sua conta *trial* já está disponível e você tem 14 dias para experimentar gratuitamente as vantagens desta plataforma escalável e automática. Acesse agora mesmo o **Painel de Controle** e crie seu ambiente!

Endereço: http://app.jelasticlw.com.br/
Login: evaldoj
Senha: LZrJT

Figura 12.18: E-mail de boas vindas do Jelastic

Clique no link do e-mail ou acesse o endereço https://app.jelasticlw.com.br/ e informe o login e senha que recebeu no e-mail:

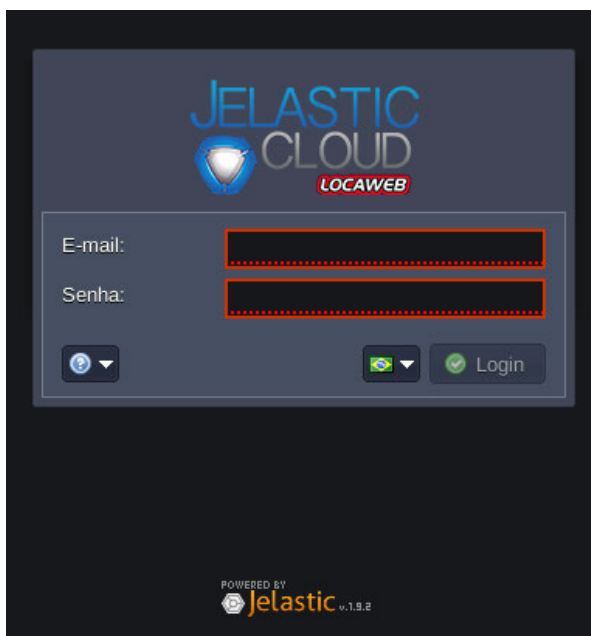



Figura 12.19: Tela de login do Jelastic

Nesta tela, você será recebido pelo assistente do Jelastic:

Figura 12.20: O assistente do Jelastic

Para a primeira utilização, eu recomendo seguir as instruções do assistente: basta clicar em `Vamos começar` e então clicar na opção `Criar ambiente` que fica em cima, à esquerda:

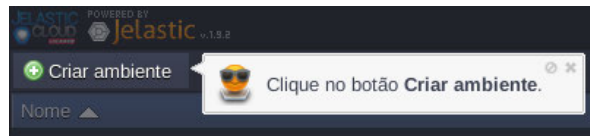



Figura 12.21: Opção para criar um novo ambiente

Nesta próxima tela temos a opção de hospedar uma aplicação Java ou PHP:

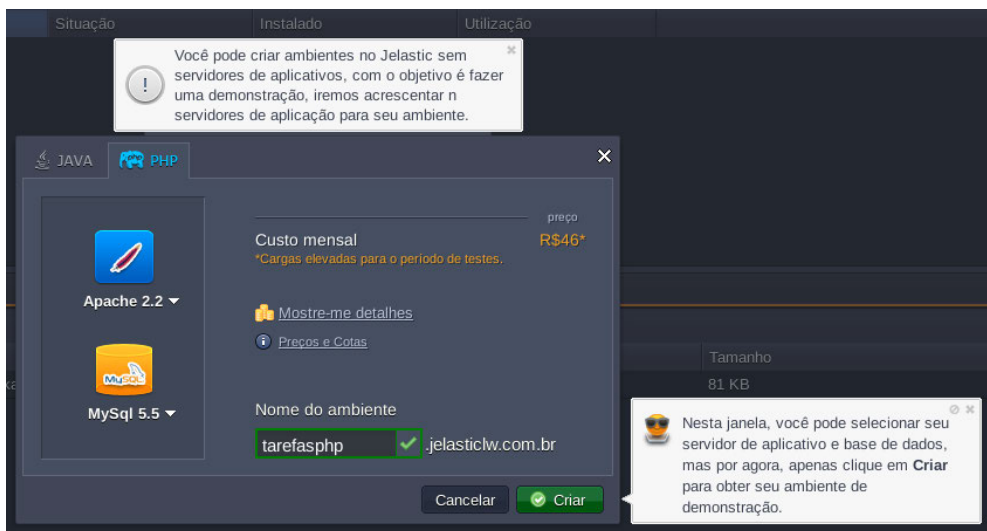



Figura 12.22: Seleção das tecnologias para rodar a aplicação

Selecione `PHP` na aba superior, `Apache` para o servidor HTTP e `MySQL` para o banco de dados. Aproveite escolher o endereço da aplicação e clique em `Criar`.

Agora é só aguardar enquanto a aplicação é criada:

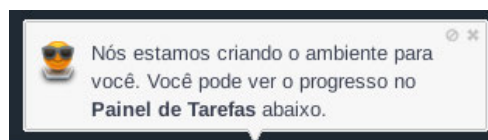



Figura 12.23: Aguarde a criação do ambiente

E assim que terminar, o assistente informa que o ambiente foi criado:

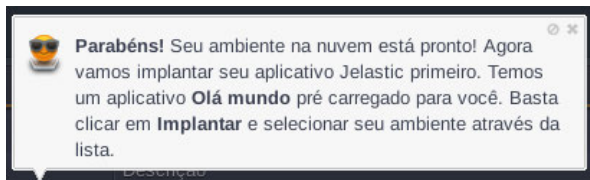



Figura 12.24: Sucesso na criação do ambiente para hospedagem

Verifique mais uma vez a sua caixa de entrada e você deverá ver um e-mail com os dados de acesso ao ambiente:

Figura 12.25: Endereço de acesso à aplicação hospedada

Você também deverá ter recebido um e-mail com a indicação de que o ambiente do MySQL foi criado e seus dados de acesso:

Figura 12.26: Dados de acesso ao MySQL

Agora é necessário acessar o endereço do banco de dados. Lá tem um PHPMyAdmin. Informe seus usuário e senha e use o PHPMyAdmin para criar um novo banco chamado `tarefas` e crie as tabelas criadas nos capítulos 6 e 10 aqui do livro.

12.9 Configurando a aplicação para o Jelastic

Após a criação das tabelas, edite o arquivo `config.php` da aplicação e coloque os dados de acesso ao MySQL do Jelastic:

```
<?php

    define("BD_SERVIDOR", "seu-mysql.jelasticlw.com.br");
    define("BD_USUARIO", "seu_usuario");
    define("BD_SENHA", "sua_senha");
    define("BD_BANCO", "tarefas");
```

Feito isso, crie um pacote `zip` da sua aplicação. Isso pode ser feito clicando com o botão direito na pasta `tarefas` e usando a opção de criar um arquivo `zip`. Este passo é importante pois no Jelastic esta é a maneira padrão de enviar a aplicação.

12.10 Enviando a aplicação para o Jelastic

Após fazer o pacote `zip` volte ao painel do Jelastic e clique na opção **Upload** que está na aba **Gerenciador de Instalação**:

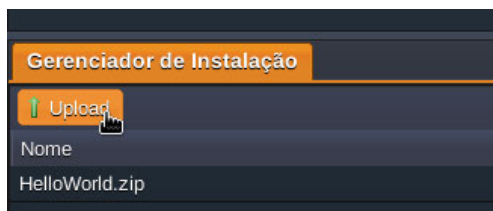



Figura 12.27: Opção de Upload da aplicação

Na próxima janela selecione o pacote `zip` que você criou e dê uma descrição para esta versão do pacote:

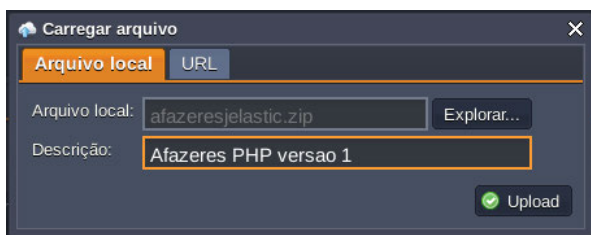



Figura 12.28: Selecione o zip e dê uma descrição para o pacote

Um bom nome ajuda a identificar o pacote quando você tiver mais de uma versão cadastrada.

O próximo passo é usar a opção de enviar o pacote para produção usando o ícone que está na lista com os nomes dos pacotes:

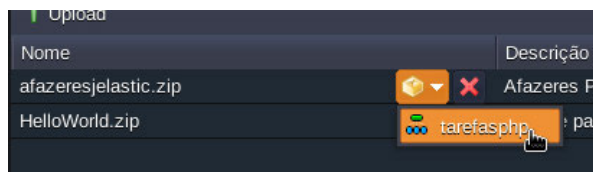



Figura 12.29: Selecione o ambiente para enviar a aplicação

Você será questionado sobre onde fazer a instalação do pacote. Deixa em branco para instalar na raiz do endereço e clique em `instalar`:

Figura 12.30: Deixe em branco para instalar na raiz do endereço

Agora aguarde enquanto a aplicação é implantada:

Figura 12.31: Aguarde a implantação

Após a conclusão você pode clicar no ícone para ir para o endereço da aplicação:

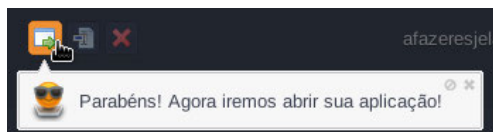



Figura 12.32: Aplicação instalada, basta acessar!

No meu caso a aplicação ficou em http://tarefasphp.jelasticlw.com.br/tarefas.php. Ao acessar a sua, não se esqueça de colocar o nome do arquivo `tarefas.php` no endereço.

E é isso, sua aplicação estará instalada no Jelastic.

12.11 Resumo

Neste capítulo foram apresentadas duas maneiras de hospedagem de aplicações PHP: uma usando o FTP e CPanel da Hostinger, que também são usados por diversos outros servidores, e outra usando o Jelastic da Locaweb que tem opções bem interessantes para configuração do servidor.

12.12 Desafios

Achou que não teríamos desafios neste capítulo? Aqui estão eles!

- Faça a hospedagem da aplicação dos contatos na Hostinger.
- Faça a hospedagem da aplicação do estacionamento no Jelastic.
- Procure mais uma opção de hospedagem PHP e MySQL e faça a hospedagem do sistema de tarefas.

Capítulo 13

Programando com orientação a objetos

No capítulo de envio de e-mails usamos a biblioteca PHPMailer através da criação de um **objeto** na variável `$email`. Este objeto era composto por diversas variáveis e funções, tudo embutido em apenas uma variável, quase como um array, com a diferença de que um array não tem funções, apenas índices com valores.

Esta é uma forma bem interessante de se desenvolver aplicações, pois podemos pensar não apenas no fluxo da aplicação, mas também nas lógicas de cada parte separadamente. A biblioteca PHPMailer é interessante porque ela não tem as lógicas necessárias para o nosso gerenciamento de tarefas, tudo o que ela faz é o envio de e-mails, e se preocupa apenas com isso.

Um exemplo interessante do que pode virar código orientado a objetos é o nosso registro de tarefas, já que todas as regras e lógicas para manipular o cadastro de tarefas estão espalhadas pela aplicação em diversas funções e condicionais, enquanto poderiam estar reunidas em uma espécie de **pacote** que seria o responsável pelas

regras relacionadas às tarefas.

Este pacote recebe o nome de **classe** e uma classe pode gerar os tais **objetos**. Olhando novamente para o uso da biblioteca PHPMailer, veja como foi criado o objeto `$email`:

```
<?php
    ...

    $email = new PHPMailer();

    ...
```

Neste caso `PHPMailer` é a classe e usamos o operador `new` para criar um novo objeto dessa classe.

Podemos fazer o mesmo criando, por exemplo, uma classe `Tarefas` que conteria todas as lógicas responsáveis por trabalhar com o banco, salvar anexos e até mesmo usar a `PHPMailer` para enviar e-mails.

13.1 A classe Tarefas

Vamos então à criação da nossa classe `Tarefas`. A ideia é que esta classe contenha toda a lógica das tarefas, portanto ela vai nos ajudar a fazer uma limpeza no arquivo `banco.php` e também no arquivo `ajudantes.php`, pois tudo vai estar dentro dela.

Vamos começar pela parte mais simples, que é ir ao banco buscar as tarefas já cadastradas. Para isso, vamos criar um diretório chamado `classes` e dentro dele um arquivo chamado `Tarefas.php` (isso mesmo, com a inicial em caixa alta). Dentro desse arquivo, coloque o esboço da classe `Tarefas`:

```
<?php

// Arquivo: classes/Tarefas.php

class Tarefas
{
    public $conexao;
    public $tarefas = array();

    public function __construct($nova_conexao)
    {
```

```
        $this->conexao = $nova_conexao;
    }
}
```

Analisando o código para a classe `Tarefas` podemos perceber alguns itens interessantes, como as variáveis `$conexao` e `$tarefas`. Aliás, a partir de agora vamos usar o nome `atributos` para essas variáveis que são membros de uma classe.

Um **atributo** é um valor que define algo em uma classe. Para ficar mais fácil de entender, imagine que se estivéssemos criando uma classe `Gato` teríamos os atributos peso, cor do pelo, tamanho das garras etc. Atributos são as qualidades de uma classe.

Lá na classe PHPMailer também vimos alguns atributos como Host, Port e Username.

Outro item que chama a atenção é a função `__construct()`. Esta é uma função que é executada sempre que um novo objeto é criado usando o operador `new`. Neste caso `__construct()` recebe como parâmetro uma conexão do MySQL e guarda no atributo `$conexao` da nossa classe. Aliás, para acessar aquele atributo `$conexao` da classe, usamos o `$this` que é um nome para acessar os objetos criados a partir desta classe.

E já que já demos o nome de **atributo** às variáveis de uma classe, vamos também usar o nome correto das funções de uma classe, que é **método**.

Voltando à classe `Gato` os métodos seriam algo como `andar()`, `miar()` e `dormir()`.

13.2 Buscando tarefas dentro da classe

Agora vamos transformar a função `buscar_tarefas()` que está no arquivo `banco.php` em um **método** da classe Tarefas. Para isso podemos recortar a função do arquivo `banco.php` e colar logo depois do método `__construct()` da classe Tarefa:

```
<?php
    ...

    public function __construct()
    {
        ...
    }
```

```
    public function buscar_tarefas()
    {
        $sqlBusca = 'SELECT * FROM tarefas';
        $resultado = mysqli_query($this->conexao, $sqlBusca);

        $this->tarefas = array();

        while ($tarefa = mysqli_fetch_assoc($resultado)) {
            $this->tarefas[] = $tarefa;
        }
    }
```

Pequenas alterações foram feitas no método `buscar_tarefas()`. Veja que ele não recebe mais a conexão MySQL, pois esta conexão já é informada na hora de criar um novo objeto. A variável `$tarefas` virou o atributo `$this->tarefas` e não é mais retornada no final, pois já fica dentro do atributo `$this->tarefas`.

Como já temos uma função transportada para a classe, vamos alterar a aplicação para que use a classe em vez da antiga função. No arquivo `tarefas.php` adicione um `include` para o arquivo da classe, não deixando de colocar o caminho dele dentro da pasta `classes`:

```
<?php

include "config.php";
include "banco.php";
include "ajudantes.php";
include "classes/Tarefas.php";

...
```

Logo abaixo do `include` já podemos criar um objeto `$tarefas` usando a classe `Tarefas`:

```
<?php
    ...

    $tarefas = new Tarefas($conexao);
```

Perceba que estamos passando a variável `$conexao` na criação do objeto. Quando criamos um objeto novo, o PHP roda o método `__construct()` que

neste caso recebe a conexão e a coloca no atributo `$this->conexao`. Por isso não precisamos mais ficar passando a conexão para o método `buscar_tarefas()`, já que ele usa o atributo do objeto.

Agora, no final do arquivo `tarefas.php` faça a chamada do método `buscar_tarefas()`. Isso vai preencher o atributo `tarefas` da classe com as tarefas que estão no banco:

```
<?php
    ...

    $tarefas->buscar_tarefas();

    include "template.php";
```

E no arquivo `tabela.php`, onde é exibida a lista de tarefas, altere o uso da variável `$lista_tarefas` pelo atributo `$tarefas` do objeto `$tarefas`:

```
...

</tr>
<?php foreach ($tarefas->tarefas as $tarefa) : ?>
    <tr>

    ...
```

Agora já podemos acessar a aplicação novamente e tudo deverá estar funcional. Visualmente, nada mudou, mas agora temos uma classe para centralizar as regras para as tarefas.

13.3 Buscando apenas uma tarefa dentro da classe

Vamos mover a função `buscar_tarefa()` para um método da classe `Tarefas`. Para isso, recorte essa função do arquivo `banco.php` e cole dentro da classe `Tarefas`, logo abaixo do método `buscar_tarefas()`:

```
<?php
    class Tarefas
    {
        ...
```

```
        public function buscar_tarefas()
        {
            ...
        }

        function buscar_tarefa($id)
        {
            $sqlBusca = 'SELECT * FROM tarefas WHERE id = ' . $id;
            $resultado = mysqli_query($this->conexao, $sqlBusca);

            $this->tarefa = mysqli_fetch_assoc($resultado);
        }
    }
```

Veja que as mudanças para este método também são pequenas, pois estamos usando a conexão que está no atributo `$this->conexao` e o resultado está sendo guardado no atributo `$this->tarefa`. Aliás, o atributo `$this->tarefa` ainda não existe, então vamos adicioná-lo logo após o atributo `$this->tarefas`:

```
<?php
    class Tarefas
    {
        public $tarefas = array();
        public $tarefa;
        public $conexao;

        ...
    }
```

Agora podemos alterar o arquivo `editar.php` para usar a classe `Tarefa` e o método `buscar_tarefa()`. Para isso devemos incluir o arquivo `classes/Tarefas.php` e criar o objeto `$tarefas` logo no início do arquivo `editar.php`:

```
<?php

include "config.php";
include "banco.php";
include "ajudantes.php";
include "classes/Tarefas.php";
```

```
$tarefas = new Tarefas($conexao);

...
```

E no final do arquivo trocamos o uso da função `buscar_tarefa()` pelo método com o mesmo nome:

```
<?php
...

$tarefas->buscar_tarefa($_GET['id']);
$tarefa = $tarefas->tarefa;

$tarefa['nome'] = (isset($_POST['nome'])) ?
    $_POST['nome'] : $tarefa['nome'];
...
```

Com isso, a aplicação já volta a funcionar com mais uma função transformada em método da classe `Tarefas`.

13.4 Gravando e editando tarefas dentro da classe

Vamos mover também a gravação de novas tarefas e a edição de tarefas para dentro da classe Tarefas. Aqui faremos basicamente os mesmos passos já feitos para mover a busca de tarefas para dentro da classe. Comece movendo as funções `gravar_tarefa()` e `editar_tarefa()` para métodos da classe, depois remova o parâmetro `$conexao` e, por último, troque o uso da variável `$conexao` pelo atributo `$this->conexao`. O método `gravar_tarefa()` fica assim:

```
<?php
    // Arquivo classes/Tarefas.php
    ...

    class Tarefas
    {
        ...

        public function gravar_tarefa($tarefa)
        {
            $sqlGravar = "
                INSERT INTO tarefas
```

```
                (nome, descricao, prioridade, prazo, concluida)
                VALUES
                (
                    '{$tarefa['nome']}',
                    '{$tarefa['descricao']}',
                    {$tarefa['prioridade']},
                    '{$tarefa['prazo']}',
                    {$tarefa['concluida']}
                )
            ";

            mysqli_query($this->conexao, $sqlGravar);
        }
    ...
```

E o método `editar_tarefa()` fica assim:

```
<?php
    // Arquivo classes/Tarefas.php
    ...

    class Tarefas
    {
        ...

        public function editar_tarefa($tarefa)
        {
            $sqlEditar = "
                UPDATE tarefas SET
                    nome = '{$tarefa['nome']}',
                    descricao = '{$tarefa['descricao']}',
                    prioridade = {$tarefa['prioridade']},
                    prazo = '{$tarefa['prazo']}',
                    concluida = {$tarefa['concluida']}
                WHERE id = {$tarefa['id']}
            ";

            mysqli_query($this->conexao, $sqlEditar);
        }

    ...
```

Depois, basta alterar o uso das funções `gravar_tarefa()` e `editar_tarefa()` pelos métodos de mesmo nome. No arquivo `tarefas.php`:

```
<?php
    ...

    if (! $tem_erros) {
        $tarefas->gravar_tarefa($tarefa);

        if (isset($_POST['lembrete']) && $_POST['lembrete'] == '1') {

        ...
```

E no arquivo `editar.php`:

```
<?php
    ...

    if (! $tem_erros) {
        $tarefas->editar_tarefa($tarefa);

        if (isset($_POST['lembrete']) && $_POST['lembrete'] == '1') {

        ...
```

Com isso, temos mais duas funções transformadas em métodos. Perceba que o arquivo `banco.php` está esvaziando e agora tem apenas a conexao com o MySQL e mais três funções (`buscar_anexos()`, `gravar_anexo()` e `remover_tarefa()`). Dessa forma, estamos movendo toda a lógica relacionada às tarefas para a classe `Tarefas` e isso é muito bom, porque fica fácil de encontrar as regras e mais organizado.

13.5 Usando o MySQLi orientado a objetos

Até agora usamos diversas funções `mysqli_` e uma coisa que incomoda um pouco é ter que ficar passando a variável com a conexão para cada função que precisamos usar. Seria bem mais interessante se a própria conexão tivesse maneiras de realizar os comandos SQL, certo?

Por isso o **MySQLi** possui uma forma orientada a objetos de trabalhar que é bem interessante e não é difícil de aplicar na nossa aplicação.

Lembra da função `mysqli_connect()` usada no arquivo `banco.php`? Ela retorna a conexão que usamos em todas as outras funções `mysqli_` até agora. Esta linha pode ser substituída pela criação de um objeto da classe `mysqli` e este objeto possui os métodos necessários para trabalharmos com o banco de dados.

Para criar o objeto, precisamos do operador `new` e vamos usar a classe `mysqli`. Então, no arquivo `banco.php` troque a linha que cria a variável `$conexao` pela criação do objeto `$mysqli`:

```
<?php
$mysqli = new mysqli(BD_SERVIDOR, BD_USUARIO, BD_SENHA, BD_BANCO);

if ($mysql->connect_errno) {
    echo "Problemas para conectar no banco. Verifique os dados!";
    die();
}

...
```

Perceba que a verificação do erro de conexão já é feita usando um atributo do objeto `$mysqli` e não mais a função `mysqli_connect_errno()`.

Agora, altere os locais que criam um objeto da classe `Tarefas` para receberem o objeto `$mysqli`. Isso acontece nos arquivos `tarefas.php`, `editar.php` e `tarefa.php`.

Veja como fica no arquivo `tarefas.php`:

```
<?php
    ...

    $tarefas = new Tarefas($mysqli);

    ...
```

É claro que isso vai refletir nos nossos métodos da classe `Tarefas` que usam as funções `mysqli_`. Então, para acertar a classe `Tarefas` vamos começar mudando o atributo `$conexao` para `mysqli`:

```
<?php

class Tarefas
{
```

```
    public $tarefas = array();
    public $tarefa;
    public $mysqli;

    ...
```

Altere também o construtor da classe `Tarefas` para usar o objeto `$mysqli`:

```
<?php
    ...

    public function __construct($novo_mysqli)
    {
        $this->mysqli = $novo_mysqli;
    }

    ...
```

E por fim, precisamos apenas trocar o uso da função `mysqli_query()` pelo método `query()` do objeto `$mysqli`. Veja como fica o método `buscar_tarefa()`, por exemplo:

```
<?php
    ...

    function buscar_tarefa($id)
    {
        $sqlBusca = 'SELECT * FROM tarefas WHERE id = ' . $id;
        $resultado = $this->mysqli->query($sqlBusca);

        $this->tarefa = mysqli_fetch_assoc($resultado);
    }
```

Agora o resultado é obtido usando `$this->mysqli->query()`, pois o `mysqli` é um objeto que é um atributo da classe **Tarefas**. Perceba que não mudamos o uso da função `mysqli_fetch_assoc()`, já que ela continua recebendo um resultado do método `$mysqli->query()`.

Altere os outros métodos da classe `Tarefas` para usar o método `this->mysqli->query()` no lugar da função `mysqli_query()`.

Antes de executar a aplicação também é necessário alterar as funções `remover_tarefa()`, `gravar_anexo()` e `buscar_anexos()` que estão no arquivo `banco.php` e que ainda não transformamos em métodos da classe `Tarefas`.

No caso, estas funções ainda recebem como parâmetro a conexão, mas agora vão receber o objeto `$mysqli`. Veja como fica a função `remover_tarefa()`:

```
<?php

function remover_tarefa($mysqli, $id)
{
    $sqlRemover = "DELETE FROM tarefas WHERE id = {$id}";

    $mysqli->query($sqlRemover);
}
```

Nesta função mudamos o parâmetro para `$mysqli` e removemos a função `mysqli_query()`, trocando seu uso pelo método `$mysqli->query()`.

Já as funções `gravar_anexo()` e `buscar_anexos()` ficam assim:

```
<?php

function gravar_anexo($mysqli, $anexo)
{
    $sqlGravar = "INSERT INTO ...  ";

    $mysqli->query($sqlGravar);
}

function buscar_anexos($mysqli, $tarefa_id)
{
    $sqlBusca = "SELECT * FROM anexos WHERE tarefa_id = {$tarefa_id}";
    $resultado = $mysqli->query($sqlBusca);

    ...
}
```

Deixei em destaque apenas as linhas mais importantes e que sofreram as mudanças para usar o objeto `$mysqli`. O restante das funções funciona da mesma maneira que antes.

Estas funções são usadas nos arquivos `tarefas.php`, `editar.php` e `remover.php`. Então devemos alterar suas chamadas para passar o objeto `$mysqli` no lugar da variável `$conexao`. No arquivo `remover.php` o uso da função `remover_tarefa()` fica assim:

```
<?php

include "config.php";
include "banco.php";

remover_tarefa($mysqli, $_GET['id']);

header('Location: tarefas.php');
```

Após alterar também as chamadas das funções `gravar_anexo()` e `buscar_anexos()` a aplicação já poderá ser executada normalmente.

O uso da biblioteca MySQLi como um objeto é interessante pois deixa quase tudo dentro de um único objeto, em vez de usar várias funções, além de não ser necessário ficar passando a variável com a conexão para cada função que é chamada.

13.6 Avançando em orientação a objetos

Orientação a objetos é muito mais do que o pouco que mostrei neste livro. A ideia aqui foi fazer apenas uma introdução para incentivar o leitor a pesquisar mais sobre o assunto.

Trabalhar com orientação a objetos costuma ser o pesadelo de muitos iniciantes em programação, mas depois que se entendem os conceitos de classes, objetos e de comunicação entre os objetos e padrões de desenvolvimento como os famosos **design patterns**, percebe-se a importância desde tipo de desenvolvimento de código.

Os mais avançados em desenvolvimento orientado a objetos podem ter encontrado diversos pontos de melhoria no código da classe Tarefas, como o fato de ela ser uma espécie de entidade, um repositório e uma coleção ao mesmo tempo. Mas não fiquem impacientes! Esta foi apenas uma maneira de mostrar como a orientação a objetos pode ser útil na centralização e organização de código. A lista de e-mails desde livro pode e deve ser usada para apresentar implementações diferentes e mais avançadas.

13.7 MVC e Frameworks

Um conceito importante que infelizmente não coube neste livro é o **MVC**, que é um padrão para desenvolvimento de software que separa a aplicação em basicamente três camadas distintas, sendo que cada uma é responsável por uma função. Neste livro

não usamos estritamente o MVC, mas os exemplos foram feitos de forma a manter as responsabilidades separadas em cada arquivo. Veja que o arquivo `banco.php` sempre foi o responsável pelo acesso ao banco, até a classe `Tarefas` aparecer. Os arquivos `tarefas.php`, `editar.php` e `remover.php` foram os responsáveis por receber as requisições do navegador e então decidir o que fazer. E também temos alguns arquivos que contêm bastante HTML e servem para montar os templates da aplicação.

No MVC existem três camadas: **Model**, **View** e **Controller**.

A camada **Model** é a responsável pelas regras de negócio e, em geral, pelo acesso ao banco de dados ou outras fontes de dados. Podemos meio que fazer uma associação da Model com nosso arquivo `banco.php` ou com a classe `Tarefas`.

A camada **Controller** é responsável por receber as requisições e decidir o que deve ser feito com elas. Em geral, o **controller** usa as **models** para obter e gravar dados, e envia informações para que as **views** exibam dados para quem solicitou.

Já a camada **View** é responsável por interagir com os usuários exibindo as páginas (as **views** também podem conversar com outros sistemas através de linguagens como **XML** e **JSON**).

Existem ferramentas que implementam o MVC e facilitam e uniformizam o trabalho dos desenvolvedores. Estas ferramentas são os **frameworks**.

Em geral um **framework** já vem com diversas diretrizes para desenvolvimento, como onde colocar um controller, como fazer uma view, uniformização de acesso aos diversos bancos de dados, ajudantes para envio de e-mails, validação de formulários, camadas de segurança etc.

Existem diversos **frameworks** para PHP, cada um com seu foco e seu público. Fica aqui uma lista com alguns **frameworks** PHP:

- CodeIgniter — leve, rápido e fácil de aprender. Bom para aplicações menores.
- Laravel — uma das estrelas do mundo PHP atualmente. Usado para a produção de projetos pequenos e grandes. Sua biblioteca de banco de dados é bastante elogiada.
- CakePHP — um dos mais conhecidos e utilizados de pequenas a grandes aplicações.
- Zend Framework — **framework** oficial da Zend, cheio de recursos e um dos preferidos das empresas para grandes aplicações.

- Symfony — um dos mais robustos, bastante usado para grandes aplicações.

Experimente conhecer dois ou três **frameworks**. Tente fazer a aplicação das tarefas em cada um deles e veja como eles tratam o acesso ao banco, como fazem o upload de arquivos etc.

É sempre bom conhecer alguns **frameworks** para poder escolher na hora de criar uma nova aplicação, além de aprender com os detalhes de cada um.

13.8 Resumo

Neste capítulo foi apresentada uma pequena introdução à programação orientada a objetos. Conseguimos criar uma classe `Tarefas` que passou a servir de centralizadora para os trechos de código responsável por lidar com as tarefas no banco de dados. Também alteramos a utilização do MySQLi para sua forma orientada a objetos e centralizamos o acesso ao banco no objeto `$mysqli` em vez de usar diversas funções.

Orientação a objetos é um tópico bastante extenso e vale a pena entender mais profundamente, pois grande parte dos softwares PHP (e de outras linguagens) a utilizam para criar sistemas grandes ou pequenos de forma mais organizada e facilitando a manutenção.

Também foi feita uma pequena introdução a conceitos como **MVC** e **frameworks**, que também são tópicos que rendem boas horas de estudo e calorosas discussões entre desenvolvedores.

13.9 Desafios

Agora mais alguns desafios, dessa vez para treinar orientação a objetos.

- Transforme as funções `buscar_anexos()`, `gravar_anexo()` e `remover_tarefa()` em métodos da classe `Tarefas`.

- Mova a validação dos formulários de criação e edição de tarefas para dentro da classe `Tarefas`. Este é mais complexo, pois você deve retornar as mensagens de erro, caso necessário.

- Crie uma classe `Contatos` e transforme as funções de manipulação do banco de dados em métodos desta classe no projeto dos contatos.

- No projeto do estacionamento crie uma classe `Veiculos` que será responsável por manipular os dados dos veículos no banco de dados.
- Escolha um dos **frameworks** listados e tente recriar a aplicação das tarefas nele. Uma dica é fazer a aplicação de exemplo que em geral existe no manual de casa **framework**. Quase sempre esta aplicação de exemplo é um blog, o que envolve diversos recursos.

Capítulo 14

Proteção e ajustes

Este é um capítulo bônus que não poderia ficar de fora deste livro.

Existem diversas formas de ataques a sites e aplicações web e, quando colocamos nosso código online, corremos o risco de receber algum tipo de ataque para quebrar a segurança de nossas aplicações e roubar informações — ou mesmo para transformar nossas aplicações em robôs que enviam spam ou outros tipos de ameaças virtuais.

Um dos tipos mais comuns de ataque é a **SQL Injection**, com a qual o atacante consegue burlar o acesso ao banco de dados da aplicação e, em geral, consegue apagar todos os dados.

Nossa aplicação das tarefas está vulnerável a este tipo de ataque, pois um usuário poderia digitar um pouco de SQL em alguns de campos. Experimente cadastrar uma tarefa chamada `Assistir 'Star Wars'` e você verá que ela não será cadastrada.

Isso acontece pois montamos o código SQL usando o apóstrofo para delimitar nossos campos de texto e, quando colocamos mais um apóstrofo no nome da tarefa, o código SQL é gerado incorretamente. Isso é ruim, certo?

14.1 Protegendo-se contra SQL Injection

Para proteger nossa aplicação precisamos escapar os caracteres que podem ser usados para compor código SQL, como é o caso de aspas e apóstrofos.

Para escapar esses caracteres, devemos usar a função `mysqli_real_escape_string()` ou o método `escape_string()` da classe **mysqli**.

Veja como fica o método `gravar_tarefa()` da classe `Tarefas`, no arquivo `classes/Tarefas.php`:

```
<?php

class Tarefas
{
    ...

    public function gravar_tarefa($tarefa)
    {
        $nome = $this->mysqli->escape_string($tarefa['nome']);
        $descricao = $this->mysqli->escape_string($tarefa['descricao']);
        $prazo = $this->mysqli->escape_string($tarefa['prazo']);

        $sqlGravar = "
            INSERT INTO tarefas
            (nome, descricao, prioridade, prazo, concluida)
            VALUES
            (
                '{$nome}',
                '{$descricao}',
                {$tarefa['prioridade']},
                '{$prazo}',
                {$tarefa['concluida']}
            )
        ";

        $this->mysqli->query($sqlGravar);
    }

    ...
```

Perceba que os dados que contêm textos foram colocados em variáveis que são

os retornos do método `$mysql->escape_string()`.

Escapar um texto é algo mais ou menos assim:

O texto era isso `Assistir 'Star Wars'` e virou isso `Assistir \'Star Wars\'`. Dessa forma, o MySQL trata o apóstrofo como um apóstrofo mesmo e não como um delimitador de campos de texto.

14.2 Exibindo campos com aspas

Agora que já estamos fazendo o escape de nossos campos de texto para o banco, um novo problema aparece. Experimente adicionar uma tarefa chamada **Assistir "Star Wars"** e você verá que o cadastro acontece normalmente.

Agora tente editar esta tarefa e você verá que o campo nome aparece apenas com o texto **Assistir**. Isso está acontecendo pelo fato de o HTML gerado ser assim:

```
<input type="text" name="nome" value="Assistir "Star Wars"" />
```

Veja que as aspas antes da palavra **Star** estão finalizando o atributo `value` da tag `input`, por isso o restante do texto não aparece dentro do campo.

Para resolver este problema, precisamos transformar as aspas em caracteres especiais de aspas que o HTML entende. Isso não é a mesma coisa que o escape de strings, mas segue uma ideia semelhante para não deixar que templates HTML quebrem.

A representação de aspas para HTML é o texto `"` e em PHP existe uma função que traduz todos os caracteres especiais do HTML em suas representações HTML. Esta função é a `htmlspecialchars()`. Veja como fica para exibirmos corretamente o nome das tarefas no formulário que está no arquivo `formulario.php`:

```
<input type="text" name="nome" value="<?php
    echo htmlspecialchars($tarefa['nome']);
    ?>" />
```

Após esta alteração, o código HTML do campo será gerado assim:

```
<input type="text" name="nome" value="Assistir &quot;Star Wars&quot;" />
```

Veja que as aspas foram alteradas para `$quot;`.

14.3 Resumo

Este foi um capítulo pequeno mas extremamente importante! Nele, foram exibidos dois conceitos muito importantes do desenvolvimento web, sendo um deles a proteção contra **SQL Injection** e o outro, o tratamento da geração de HTML para evitar quebras de campos e templates.

Além do SQL Injection, existem outros tipos de ataques como **XSS** e **CSRF** e estes tópicos merecem um estudo mais aprofundado.

Uma dica legal é que, em geral, os **frameworks** tratam destes tópicos, então este é mais um motivo para pesquisar mais sobre eles.

14.4 Desafios

E aqui estão mais alguns desafios!

- Faça o tratamento contra SQL Injection em toda a aplicação das tarefas.
- Faça o tratamento da exibição dos campos de texto da aplicação das tarefas.
- Faça o tratamento contra SQL Injection e o tratamento da exibição dos campos de texto da aplicação dos contatos e também do estacionamento.

Capítulo 15

Ao infinito... E além!

E aqui chegamos ao fim desde livro, mas não ao fim dos estudos!

Para aprender a dirigir automóveis é necessário praticar por várias horas. O mesmo é válido para quem quer aprender a tocar um instrumento musical ou para quem quer dominar algum tipo de arte ou tecnologia.

É claro que com programação não é diferente. Para se tornar um bom programador é necessário praticar. E quanto mais se pratica, mais se aprende.

Invente novos problemas para resolver, pois assim você se torna o primeiro usuário de suas aplicações. Crie um organizador de tarefas, um gestor para a sua coleção de livros, uma agenda digital, um clone do Twitter etc. O importante é continuar praticando.

15.1 Onde posso buscar mais informações?

A internet é bem vasta e as vezes é complicado achar bons conteúdos, por isso deixo aqui algumas sugestões de onde buscar ajuda e procurar informações.

Para saber mais sobre os recursos do PHP visite a documentação oficial em http://php.net/docs.php. Tem também a versão em português aqui http://www.php.net/manual/pt_BR/index.php.

Um site que tem muita informação importante sobre as melhores práticas para PHP é o http://www.phptherightway.com/. Existe também uma versão em português aqui http://br.phptherightway.com/.

Participe da lista de discussão do livro. Lá você pode tirar suas dúvidas, ajudar quem está passando por problemas diferentes e ainda conhecer pessoas interessadas no mesmo assunto, o que pode turbinar seu aprendizado. A lista está neste endereço: https://groups.google.com/forum/#!forum/phpemysqlcasadocodigo

Uma boa ideia é participar também de comunidades maiores como o GUJ – http://www.guj.com.br. Lá tem pessoas conversando sobre várias linguagens e tecnologias e é possível aprender muito participando desde tipo de grupo.

Fique de olho no site da Casa do Código – http://www.casadocodigo.com.br – para saber dos últimos lançamentos.

Vale lembrar que você pode baixar todos os exemplos usados no livro. Eles estão no GitHub em https://github.com/InFog/phpmysql

Bons estudos e sucesso!